UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Conversor de formatos para programas de análisis de poblaciones.

Autor/es

DAVID RUIZ CANTABRANA

Director/es

JÓNATAN HERAS VICENTE y JUAN JOSÉ OLARTE LARREA ,

Facultad

Facultad de Ciencia y Tecnología

Titulación

Grado en Ingeniería Informática

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2016-17

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

Facultad de Ciencia y Tecnología

TRABAJO FIN DE GRADO

Grado en Ingenieria Informática

Conversor de formatos para programas de análisis de poblaciones

Alumno:

David Ruiz Cantabrana

Tutores:

Juan José Olarte Larrea

Jónatan Heras Vicente

Logroño, junio, 2017

Resumen.

El estudio de los geles de ADN con marcadores dominantes es un campo muy amplio donde existen un gran número de programas para el análisis de los mismos, este gran número es debido a que cada programa ofrece una funcionalidad distinta. El problema con el que se enfrentan los investigadores es que cada programa tiene su propio formato y estos formatos no son compatibles entre sí, por ello el objetivo de este trabajo es paliar el problema de la incompatibilidad de formatos que existen entre este gran número de programas de análisis, desarrollando una aplicación que permita convertir los archivos de un tipo de formato a otro, permitiendo aprovechar las distintas funcionalidades de los diferentes programas de análisis genético.

Abstract.

The study of DNA gels with dominant markers is a very broad field where there is a large number of programs for their analysis, this large number is because each program offers a different functionality. The problem faced by researchers is that each program has its own format and these formats are not compatible with each other, therefore, the objective of this work is to solve the problem of the incompatibility of formats that exists between a large number of analysis programs. Consequently we will develop an application that allows us to convert files from one tool to another, this would allow to take advantage of the different functionalities of the different analysis genetics programs.

Agradecimientos.

Mi más sincero agradecimiento:

- A la Universidad de La Rioja, y al Jefe de Estudios de Informática por la ayuda recibida, debido a los problemas que tuve con la asignación del TFG.
- A Juan José Olarte y a Jónathan Heras, por la ayuda recibida y por el apoyo para la realización de este proyecto.
- A mis compañeros y amigos, por la ayuda recibida durante toda la carrera.
- A mis padres y hermanos por el apoyo recibido tanto moral como económico para poder desarrollar mi formación.
- A mi familia restante por apoyarme siempre y por preocuparse por mí.
- A mi fiel compañera que tanto me ha aguantado y me ha ayudado a seguir adelante.

Contenido

1. Introducción. 1
   1.1. Contexto y Motivación. 1
   1.2. Objetivos. 2
2. Planificación. 3
   2.1. Estructura de Descomposición de Trabajo. 3
   2.2. Descripción de las tareas. 5
   2.3. Distribución temporal. 6
   2.4. Plan de comunicación. 8
   2.5. Tecnologías a utilizar. 8
   2.6. Metodología a utilizar. 9
   2.7. Planificación de riesgos. 9
   2.8. Estudio de viabilidad. 10
3. Análisis. 11
   3.1. Herramientas para el análisis de poblaciones. 11
   3.2. Alcance 13
   3.3. Análisis de requisitos. 13
   3.4. Catálogo de requisitos. 14
      3.4.1 Requisitos funcionales. 14
      3.4.2 Requisitos no funcionales. 14
4. Diseño. 16
   4.1. PopXML. 16
   4.2. Arquitectura del sistema. 17
   4.3. Interfaz gráfica. 17
5. Implementación. 19
   5.1. Transformación del fichero plano a XML. 20
      5.1.1 1ª Iteración. 23
      5.1.2 2ª Iteración. 27
      5.1.3 3ª Iteración. 32
   5.2. Creación de la aplicación. 37
      5.2.1 Transformación XML a fichero plano. 40
      5.2.2 Procedimiento de la aplicación. 42

6. Pruebas. 44
6.1. Pruebas de la aplicación en local. 44
6.1.1 GenAlEx a Geneland 44
6.1.2 GenAlEx a Genepop 45
6.1.3 Genepop a GenAlEx 47
6.2. Pruebas en el servidor. 47
7. Seguimiento y control. 50
7.1. Plan de seguimiento y control. 50
7.2. Seguimiento y Control. 50
8. Conclusiones. 52
8.1. Lecciones aprendidas. 52
8.2. Propuesta de mejoras. 53
Bibliografía. 54

1. Introducción.

En esta sección se detallarán una descripción del problema, cómo vamos a afrontarlo y cuál es el objetivo del mismo.

1.1. Contexto y Motivación.

Los marcadores moleculares son segmentos de ADN que se consideran como puntos de referencia para el análisis del genoma. Estos marcadores tienen entre otras aplicaciones la estimación de distancias genéticas entre poblaciones, identificación y distinción de variedades, establecimiento de líneas de parentesco, y localización e identificación de genes cualitativos.

En el ámbito de la investigación sobre los análisis genéticos de poblaciones, la tarea de transformación manual de huellas dactilares de ADN es muy costosa, puesto que se necesita tiempo para hacerla y es muy propensa a errores, especialmente cuando el investigador trata con un gran número de muestras.

En la actualidad existe una gran variedad de herramientas software utilizadas para el análisis genético de poblaciones con marcadores moleculares. Sin embargo, no existe un estándar de formato de entrada para interactuar con estas herramientas.

Por ello se añade el problema de la incompatibilidad de formatos que existen entre las distintas herramientas para el análisis genético de poblaciones, dado que el investigador necesita trabajar con varias herramientas distintas, ya que cada herramienta proporciona una funcionalidad distinta.

Una solución parcial a la falta de formato común fue proporcionada por GelJ, una herramienta que a partir de imágenes de geles de ADN genera la entrada de las distintas herramientas de análisis genético con el formato adecuado. Para esta múltiple variedad de formatos, GelJ usa un XML intermedio llamado PopXML para posteriormente hacer una transformación mediante XSLTs, generando los ficheros de las distintas herramientas como se muestra en la figura 1.1.

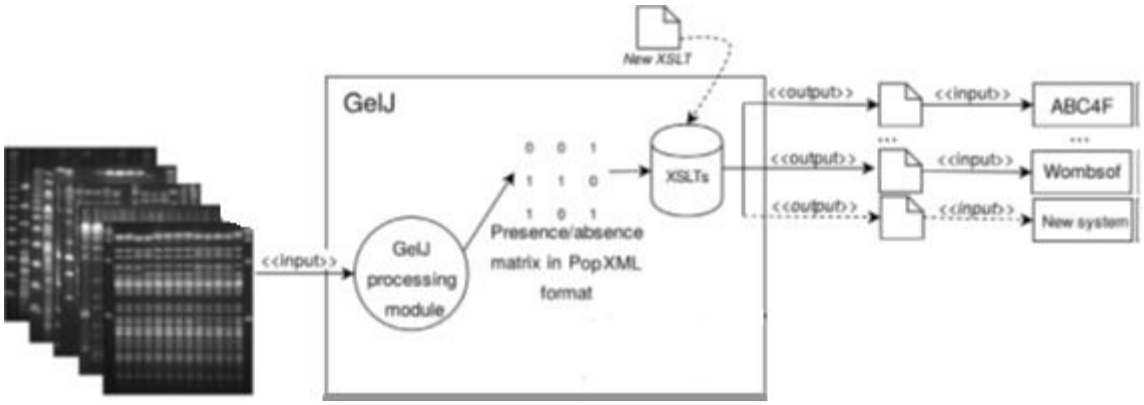

Figura 1.1. Funcionamiento de GelJ.

El problema que encuentran los investigadores es que, disponen de ficheros que tienen un formato de una herramienta concreta generada de manera independiente a GelJ, pero necesitan usar herramientas (con formatos distintos) para realizar otros análisis.

Este problema de incompatibilidad entre formatos, ha motivado la realización de un servicio en el cual se pueda lograr la interoperabilidad entre distintas herramientas usando PopXML como lenguaje intermedio.

Toda la información que se precisaba para realizar este trabajo ha sido proporcionada por Jónathan Heras Vicente que, además de ser uno de los tutores del proyecto, ha actuado con el rol de cliente, y Juan José Olarte Larrea como tutor y director del proyecto.

1.2. Objetivos.

Para abordar el tema que se ha presentado anteriormente se desea desarrollar una herramienta que solucione el problema de la interoperabilidad de las distintas herramientas de análisis genético de poblaciones.

Para ello, la herramienta a realizar debe permitir transformar ficheros planos de un formato de programa a otro formato de programa de análisis de poblaciones.

2. Planificación.

En esta sección se detalla la planificación del trabajo, el cual comienza el día 20 de febrero y está dividida por tareas.

2.1. Estructura de Descomposición de Trabajo.

Para la realización de este trabajo se ha realizado una descomposición en tareas que podemos observar en la EDT de la figura 2.1.

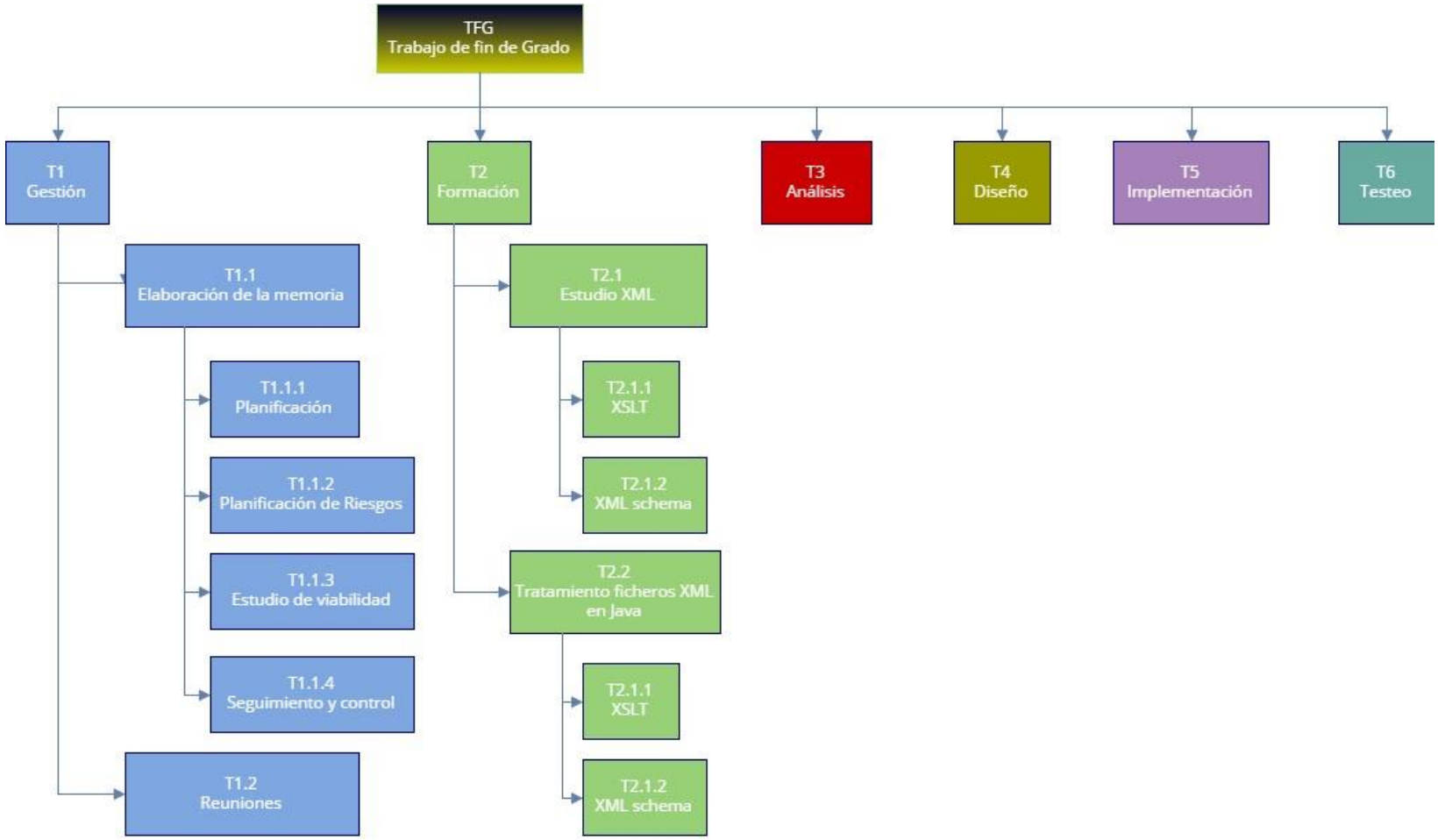

Figura 2.1. EDT.

2.2. Descripción de las tareas.

En las siguientes tablas se van a describir las diferentes tareas que se han plasmado en la anterior figura de la EDT, en la cual tenemos la tabla de Gestión (tabla 2.1), la tabla de Estudio (tabla 2.2), la tabla de Análisis (tabla 2.3), la tabla de Diseño (tabla 2.4), la tabla de Implementación (tabla 2.5) y la tabla de Testeo (tabla 2.6).

T1 - Gestión		
T1.1 – Elaboración de la memoria		
T1.1.1 – Planificación.	Creación de la planificación que se va a llevar a cabo para la realización de este trabajo.	3H
T1.1.2 – Planificación de Riesgos.	Estudio sobre los posibles riesgos del trabajo, para prevenirlos, y como solucionarlos.	2H
T1.1.3 – Estudio de viabilidad.	Estudio sobre el cual se puede decidir si el trabajo se desarrollara completamente.	3H
T1.1.4 – Seguimiento y control.	Seguimiento del trabajando anotando el tiempo empleado en cada tarea, las decisiones que se han tomado, los problemas que han ido surgiendo y cualquier cambio que afecte al alcance del mismo.	30H
T1.2 – Reuniones.	Reuniones que se van a llevar a cabo de forma periódica con el cliente y con el tutor.	12H
TOTAL		50H

Tabla 2.1. Gestión.

T2 - Formación		
T2.1 – Estudio XML		
T2.1.1 – XSLT.	Aprender las diferentes etiquetas para poder comprender los diferentes cambios que van a hacer los ficheros XML.	5H
T2.1.2 – XML schema.	Estudiar sobre las etiquetas para validar los diferentes archivos XML.	5H
T2.2 – Tratamiento ficheros XML en Java		
T2.2.1 – XSLT.	Entender el proceso de aplicar un XSLT a un fichero XML.	15H
T2.2.2 – XML schema.	Entender el proceso que hay que seguir para poder transformar un fichero XML, para que pueda seguir un XML schema.	10H
TOTAL		35H

Tabla 2.2. Formación.

T3 – Análisis		
T3– Análisis de Requisitos.	Especificar y recopilar la información que tiene que contener el producto final.	15H
TOTAL		15H

Tabla 2.3. Análisis.

T4 – Diseño		
T4– Diseño.	Definir un modelo que satisfaga los requisitos pedidos por el cliente, decidiendo las tecnologías más apropiadas para utilizar en la implementación.	35H
TOTAL		35H

Tabla 2.4. Diseño.

T5 – Implementación		
T5 – Implementación.	Utilizar las herramientas para desarrollar el código para la aplicación.	135H
TOTAL		135H

Tabla 2.5.Implementación.

T6 – Testeo		
T6 – Testeo.	Probar el funcionamiento de la aplicación, comprobando que cumple todos los requisitos.	30H
TOTAL		30H

Tabla 2.6. Testeo.

En total, el proyecto consta de 202 + 35 + 63 = 300 horas.

2.3. Distribución temporal.

El trabajo se compone de 300 horas, las cuales estarán distribuidas en 18 semanas hasta la fecha de entrega de la memoria. En cada semana se le dedicará al proyecto unas 20-25 horas, y 2 horas para realizar la memoria, estas horas pueden variar un poco en función de la carga de trabajo que tenga de las asignaturas de la carrera y de los días y semanas festivas. La distribución temporal quedaría como se muestra en la tabla 2.7.

Tabla 2.7. Distribución temporal.

Para tener un mayor control sobre el proceso del trabajo, se han propuesto una serie de hitos dentro de cada fase anteriormente expuesta. En la tabla 2.8 se pueden observar los hitos más importantes.

Hitos	Semanas									
	S1	S2	S3	S5	S8	S9	S14	S16	S17	S18
	20-25 Febrero	27-3 Marzo	6-10 Marzo	20-24 Marzo	10-14 Marzo	17-21 Marzo	22-26 Mayo	5-9 Junio	12-16 Junio	21-23 Junio
Inicio de TFG	♦									
Reunión Inicio	♦									
Planificación		♦								
Análisis			♦							
Estudio				♦						
Reunión				♦						
Diseño					♦					
Implementación						♦				

Reunión						♦				
Reunión Implementación							♦			
Pruebas								♦		
Memoria									♦	
Seguimiento y control									♦	
Reunión final TFG									♦	
Deposito										♦

Tabla 2.8. Diagrama de hitos.

2.4. Plan de comunicación.

Para que haya buena comunicación entre todos los integrantes de este proyecto, ya sea con el cliente o el director del mismo, se van a utilizar los siguientes sistemas de comunicación:

- Correo electrónico: este sistema de comunicación se utilizara para resolver pequeñas dudas o incidencias, informar sobre el estado del proyecto y para concretar las reuniones, tanto con el cliente como con los tutores.
- Reuniones: se utilizara este sistema de comunicación para tratar los aspectos más importantes del proyecto, ya sea con el cliente para mostrar el avance del mismo u obtener información sobre él, o con los tutores para mostrar el avance y los cambios que hay que hacer.

2.5. Tecnologías a utilizar.

Para desarrollar este proyecto, el cliente no puso ninguna restricción en cuanto al lenguaje para realizar la misma, por ello se ha optado por elegir el lenguaje de programación Java, puesto que es fácil de utilizar y ya conocido por el alumno.

Netbeans IDE será utilizado para desarrollar la herramienta, se ha decantado por este IDE debido al control del alumno sobre este y al ofrecer ayudas de detección de errores, autocompletado de código y atajos de teclado cómodos para el desarrollo de la aplicación.

Se utilizara Oxygen XML para procesar los distintos archivos XML, XML Schema y XSLTs.

Para la realización del prototipo de la página web se utilizara la herramienta de moqups.com.

Para realizar copias de seguridad y tener los archivos siempre guardados por si ocurre algún imprevisto se ha utilizado la herramienta de Git, para tener toda la implementación guardada.

2.6. Metodología a utilizar.

Debido a que no está claro cuantas transformaciones de formato dará tiempo a implementar en el proceso para la conversión de archivos se ha determinado elegir un modelo iterativo-incremental. Cabe destacar que no se llevará a cabo como se define el modelo, una vez hecha la primera iteración con el análisis, diseño, implementación y pruebas, a partir de ahí las siguientes iteraciones serán con la implementación y las pruebas, dado que el análisis y el diseño será general para todas las iteraciones, esto es debido a que el único cambio que tiene cada iteración, es la forma de implementar la transformación de cada herramienta de análisis genético de poblaciones.

2.7. Planificación de riesgos.

A continuación se enumeran los posibles riesgos que el trabajo puede tener.

1. Error en la gestión del TFG.
    - Prevención: Intentar tener claro cuál es el objetivo del TFG y trabajar sobre ello.
    - Contingencia: Consultar cualquier duda con los tutores del TFG, para poder resolverlo cuanto antes.
2. Conocimiento leve sobre las tecnologías a utilizar.
    - Prevención: Consultar blogs, o la propia documentación de la herramienta.
    - Contingencia: Hacer un estudio básico para poder empezar a realizar el trabajo e ir aumentando este estudio a medida que avanza el trabajo para poder afrontar el problema.
3. Escasa información sobre la herramienta a utilizar.
    - Prevención: Consultar previamente si la herramienta puede hacer el trabajo para la cual se va a necesitar.
    - Contingencia: Utilizar otra herramienta, o hacer un estudio de la misma intentando conseguir el objetivo para la cual se utilizaba.
4. Pérdida de trabajo.
    - Prevención: Hacer copias de seguridad periódicas para poder almacenar el progreso, utilizando un sistema de control de versiones.
    - Contingencia: Elegir la copia de seguridad más antigua para poder restaurarla.

5. Problemas con el desarrollo de la aplicación.
    - Prevención: Diseñar el producto teniendo en cuenta las limitaciones de tiempo, formación y tecnológicas.
    - Contingencia: Reducir el alcance del proyecto y pedir ayuda al tutor del TFG.

2.8. Estudio de viabilidad.

Vemos conveniente hacer un estudio de viabilidad de este proyecto ya que no hay nada igual, ni tampoco se ha desarrollado una herramienta similar.

El cliente necesita una herramienta para poder solucionar el problema que anteriormente se describe en este documento. Dicha herramienta necesita un desarrollo a medida, dado que no existe nada ya desarrollado que satisfaga sus necesidades.

Los recursos y herramientas para llevar a cabo este proyecto están a mi alcance y no tienen coste, el tiempo que se dispone para la realización de este proyecto es igual a la carga de trabajo que se asocia a un TFG (300 Horas).

Queremos ver si el alcance de este proyecto podrá llevarse a cabo con la información que se plasma en este documento, por ello cuando se acabe el producto se espera que haya un prototipo funcional del cual se pueda implementar una versión más robusta y con más opciones de configuración.

Dentro del plazo de tiempo que se tiene estimado (18 semanas), se entregará un prototipo funcional, pero no se sabe con certeza hasta qué punto se podrá hacer la implementación de todas las transformaciones de herramientas, pero con la información sobre el estudio de las herramientas y sobre las tecnologías a utilizar se cree viable la realización de este trabajo.

Con el conocimiento adquirido durante el estudio del grado y el conocimiento adquirido mediante el estudio de las tecnologías a utilizar, se estima que se lograran los objetivos marcados por el cliente.

3. Análisis.

Una vez hecha la introducción en la cual se presenta el trabajo a realizar, ahora se analizarán con detalle los principales requisitos que se quieren lograr.

3.1. Herramientas para el análisis de poblaciones.

Como hemos venido comentando a lo largo de este documento el problema de la interoperabilidad viene causado por la variedad de formatos que tienen las herramientas que hay para el estudio de análisis de poblaciones.

Todas estas herramientas trabajan con ficheros de texto plano donde se codifican matrices binarias, pero cada una utiliza un formato, y la información incluida en estos ficheros varia de unas a otras. Esta información varía dependiendo de la funcionalidad de la herramienta con la que se está trabajando.

Esta información, que se incluye en algunos ficheros, se muestra en la Tabla 3.1, pueden ser el número de poblaciones, el número de Loci, número de individuos, nombre de Loci, de poblaciones, de individuos y el número de individuos por población.

A continuación en la tabla 3.1 se muestra la lista de herramientas, que toman matrices binarias llamadas de presencia / ausencia como entrada.

Tool	Year	Version	Representation of presence/absence matrix	Additional information in the input file
ABC4F	2008	1.0	Number of individuals observed at each locus.	Number of loci, number of populations, and individuals of each population.
Adegenet	2016	2.0.1	Matrix of 1s (presence) and 0s (absence) separated by tabs.	Names of loci, and names of individuals.
aflp-surv	2002	1.0	Matrix of 1s (presence) and 0s (absence) separated by tabs.	Number of loci, number of populations, names of loci, names of individuals, names of populations, and individuals of each population.
AFLPdat	2008	23/01/08	Matrix of 1s (presence) and 0s (absence) separated by tabs.	Names of loci, names of individuals, names of populations, and individuals of each population.
Alleles in space (AIS)	2005	1.0	Matrix of 1s (presence) and 2s (absence) separated by commas.	Number of loci, names of individuals.
Arlequin	2015	3.5	Matrix of 1s (presence) and 0s (absence) without separation.	Number of individuals, number of populations, names of individuals, and individuals of each population.
Baps	2013	6.0	Matrix of 1s (presence) and 0s (absence) separated by tabs.	None.
Bayescan	2012	2.1	Number of individuals observed at each locus.	Number of loci, number of populations, and individuals of each population.
EDENetworks	2014	2.18	Matrix of 1s (presence) and 0s (absence) separated by tabs.	Names of individuals.
Famd	2013	1.3	Matrix of 1s (presence) and 0s (absence) separated by tabs.	Names of loci, and names of individuals.
Famoz	2009	11/06/09	Matrix of 1s (presence) and 0s (absence) separated by tabs.	Number of loci, and number of individuals per population.
Gda	2008	1.1	Matrix of 1s (presence) and 0s (absence) separated by tabs.	Number of loci, number of populations, names of loci, names of populations, names of individuals, and individuals of each population.
GenAlEx	2012	6.502	Matrix of 1s (presence) and 0s (absence) separated by commas.	Number of loci, number of individuals, number of populations, number of individuals per population, names of populations, names of loci, and names of individuals.
Geneland	2012	4.0.5	Matrix of 1s (presence) and 0s (absence) separated by tabs.	None.
Geneticstudio	2014	1.5	Matrix of 1s (presence) and 0s (absence) separated by commas.	Names of loci, and names of individuals.
GenoDive	2013	2.0b23	Matrix of 1s (presence) and 0s (absence) separated by tabs.	Number of individuals, number of populations, number of loci, names of populations, names of loci, names of individuals, and individuals of each population.
Hickory	2004	1.1	Matrix of 1s (presence) and 0s (absence) separated by tabs.	Number of loci, number of populations, names of populations, names of individuals, and individuals of each population.
Mcheza	2011	1.0	Matrix of 1s (presence) and 2s (absence) separated by tabs.	Names of loci, names of individuals, and individuals of each population.
NewHybrids	2003	1.1 Beta 3	Matrix of +s (presence) and -s (absence) separated by tabs.	Number of individuals, number of loci, and names of loci.
Past	2015	1.0	Matrix of 1s (presence) and 0s (absence) separated by commas.	Names of loci, and names of individuals.
Piccalc	2012	1.0	Matrix of 1s (presence) and 0s (absence) without separation.	Number of individuals, and names of individuals.
Popgene	2000	1.32	Matrix of 1s (presence) and 0s (absence) without separation.	Number of populations, number of loci, names of loci, names of populations, and individuals of each population.
Sambada	2015	0.5.1	Matrix of 1s (presence) and 0s (absence) separated by tabs.	Names of loci, names of individuals, number of loci, and number of individuals.
Spagedi	2015	1.5	Matrix of 1s (presence) and 0s (absence) separated by tabs.	Number of individuals, number of populations, number of loci, names of loci, names of individuals, names of populations, and individuals of each population.
Structure	2012	2.3.4	Matrix of 1s (presence) and 0s (absence) separated by tabs.	Names of individuals, number of individuals, and number of loci.
Tess	2015	3	Matrix of 1s (presence) and 0s (absence) without separation.	None.
TFPGA	1997	1.3	Matrix of 1s (presence) and 2s (absence) separated by commas.	Individuals of each population.
Treecon	1998	1.3b	Matrix of 1s (presence) and 0s (absence) without separation.	Number of loci, and names of individuals.
Wombsoft	2007	1.0	Matrix of 1s (presence) and 0s (absence) separated by tabs.	Names of individuals.

*Tabla 3.1. Lista de herramientas. **

** Obtenido de: https://www.journals.elsevier.com/computer-methods-and-programs-in-biomedicine*

3.2. Alcance

En la reunión con el cliente se vio que había 32 herramientas distintas para el estudio de análisis genético de poblaciones (ver tabla 3.1), por ello se acordó que implementar el conversor de las 32 herramientas podría llegar a ser muy costoso y por lo tanto sobrepasar el límite de horas establecido para la realización del TFG.

Por lo que el cliente dio una lista de 3 herramientas para implementar en la conversión, que son: GenAlEX, Genepop y Geneland. Así que los ficheros que podrá recibir nuestra aplicación a desarrollar son solo de estas 3 herramientas, pero la salida que ofrecerá nuestra aplicación será de cualquiera de las otras 32 herramientas de análisis de genético de poblaciones.

En una primera versión de la aplicación se implementará la conversión de las 3 herramientas comentadas anteriormente, si se alcanza esta primera versión y es posible mejorarla, se implementará un servicio adicional para poder obtener del usuario los datos que la herramienta de origen no la proporciona y que sería de utilidad que se viese en la transformación para la herramienta de salida, o en casa contrario que se inserten de forma aleatoria.

Si algunas de estas opciones no se llevan a cabo se dejará para una posible mejora de la aplicación posteriormente.

3.3. Análisis de requisitos.

Tras varias reuniones con el cliente, se han identificado los siguientes requisitos.

La aplicación que se va a desarrollar solucionará el problema anteriormente descrito, poder tener interoperabilidad entre las distintas herramientas de análisis genético de poblaciones con marcadores dominantes. Para ello la aplicación será Web, para dar servicio a todas las personas que quieran trabajar con las distintas herramientas de análisis genético de poblaciones, sin necesidad de instalar ningún programa adicional.

La aplicación Web que se desarrollará tiene que tener un formulario en el cual reciba 3 parámetros: fichero plano de la herramienta, nombre de la herramienta de origen y nombre de la herramienta a la cual hacer la conversión. La salida debe ser un fichero plano que sea compatible con la herramienta de análisis genético seleccionada.

Para el proceso de conversión de un fichero a otro, se va a seguir el siguiente procedimiento que está basado en PoPXML:

- Transformar el archivo plano de la herramienta seleccionada en el formulario, a un archivo XML siguiendo el esquema de PopXML.

- Aplicar la hoja de estilo XSLTs de la herramienta de salida seleccionada, al XML anteriormente creado, que nos devuelve un fichero plano compatible con la herramienta seleccionada en el formulario de la aplicación Web.

3.4. Catálogo de requisitos.

Se mostraran los distintos requisitos funcionales y no funcionales que tendrá la aplicación a desarrollar.

3.4.1 Requisitos funcionales.

- La herramienta a desarrollar como se ha comentado anteriormente será Web, para poder servir a los distintos investigadores, que deseen trabajar con varias herramientas de análisis genético de poblaciones.
- La aplicación permitirá la transformación de un formato a otro.
- La aplicación permitirá que el tipo de salida sea en cualquiera de las 32 herramientas comentadas anteriormente.
- Los formatos de entrada para la conversión que tendrá la aplicación serán los siguientes:
    i. GenAlEX
    ii. Genepop
    iii. Geneland
- La aplicación a desarrollar tendrá un formulario en el cual se obtendrán 3 parámetros para la conversión de archivos:
    i. Entrada del fichero de la herramienta.
    ii. Nombre de la herramienta del fichero.
    iii. Nombre de la herramienta a la que hacer la transformación al fichero que nos da como entrada.

Si se cumplen estos requisitos el cliente ha añadido funcionalidad adicional.

- Añadir la conversión de más herramientas de análisis de población.
- Crear un formulario, en el cual permita rellenar campos que están vacíos dentro del XML.

3.4.2 Requisitos no funcionales.

- Ser capaz de añadir nuevas transformaciones de manera sencilla.
- La aplicación no tendrá ninguna contraseña ni ningún dato sensible, solo se utilizara para la interoperabilidad de las herramientas.
- La aplicación estará en inglés.

- El tiempo de respuesta de la aplicación será pequeño, a no ser de que el número de llamadas a la aplicación sea grande, en este caso la respuesta podría verse afectada y tener un tiempo de respuesta medio.
- La interfaz Web seguirá los patrones HTML y CSS, al igual que el cumplimiento del patrón MVC.
- La temática/estilo de la web seguirá la línea de la Universidad de La Rioja en cuanto a colores e imágenes de la misma.
- La aplicación Web no será Responsive debido a que la utilidad de las herramientas será por otros programas que se deben tener en el ordenador.
- La aplicación será fácil de usar e intuitiva para la persona que la utilice.

4. Diseño.

En esta sección se detallarán los aspectos más importantes de la fase de diseño del proyecto.

4.1. PopXML.

Actualmente existe una amplia variedad de herramientas software para el análisis genético de poblaciones con marcadores dominantes que presentan, entre otras funcionalidades, el cálculo de índices de diversidad genética, el estadístico F y la visualización y edición de árboles filogenéticos.

GelJ es la primera herramienta existente que llena la brecha bioinformática entre el software de procesamiento de imágenes de geles y el análisis genético de poblaciones, la reconstrucción filogenética y el software de edición de árboles.

Para ello GelJ utiliza un lenguaje XML intermedio llamado PopXML, el cual a través de las imágenes de geles de ADN extrae las matrices de presencia/ausencia y crea un XML mediante el esquema XML de PopXML, el cual se muestra en la figura 4.1 y aplicando hojas de estilo XSLTs a ese XML, se da salida a los ficheros planos compatibles con las distintas herramientas de análisis genético de poblaciones.

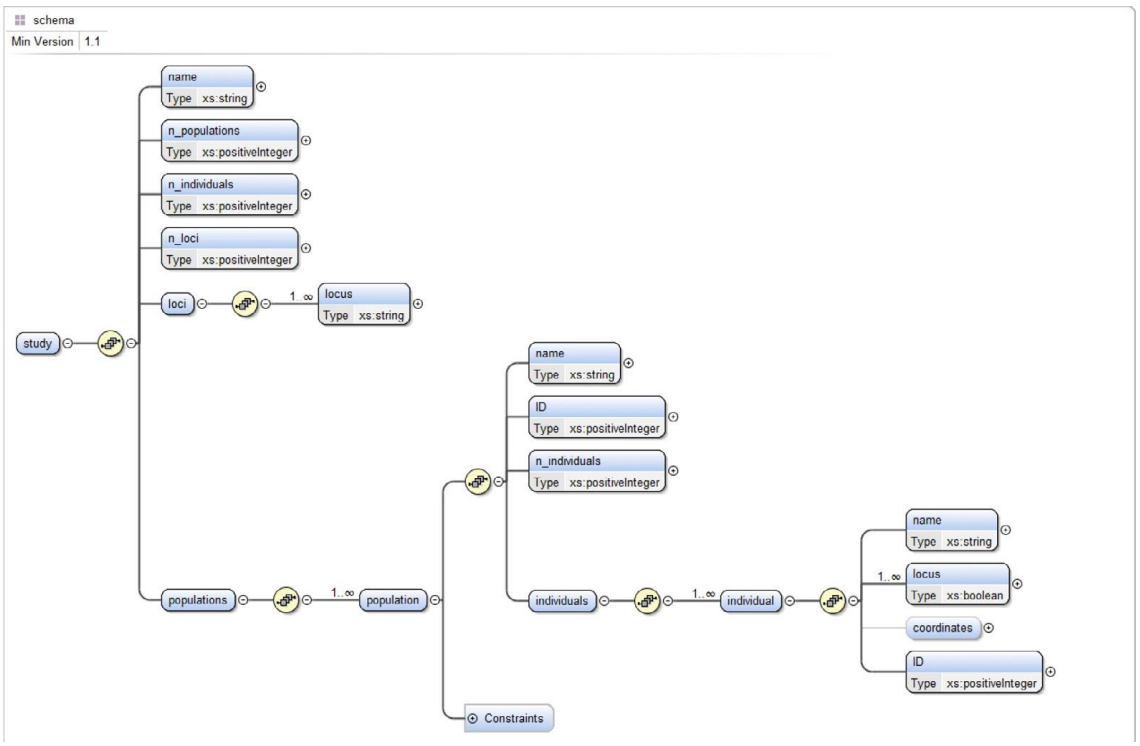

Figura 4.1. Esquema XML de PopXML

La herramienta a desarrollar tiene que hacer el proceso inverso al que hace GelJ sin tener que generar la imagen de geles de ADN, por lo que tiene que pasar del fichero plano de salida al XML intermedio, para una vez obtenido este XML poder aplicarle el XSLTs de la herramienta correspondiente con la que se quiere trabajar.

Todas las hojas de estilo XSLTs se han obtenido gracias al cliente, por lo que tendremos una lista con todas ellas para poder aplicársela a un XML y que nos devuelva la salida en la herramienta seleccionada.

4.2. Arquitectura del sistema.

La aplicación que se va a desarrollar, es una aplicación web por lo que utilizaremos el uso de una arquitectura en tres capas, concretamente, el patrón MVC (Modelo, Vista, Controlador).

Para ello la aplicación tiene que tener por lo menos una transformación que pueda cambiar de un archivo plano de una herramienta concreta a otro archivo plano de otra herramienta concreta. Para la realización es necesario poder cambiar el archivo plano de origen a un formato XML, para después aplicarle una hoja de estilo XSLT y que el archivo de salida sea del formato que se había seleccionado con anterioridad en el formulario.

4.3. Interfaz gráfica.

La elaboración de la interfaz gráfica es sencilla debido a que como ya hemos venido comentando anteriormente en este documento, es un formulario en el cual se deposita un archivo y el servicio devuelve otro, en la figura 4.2 se ve el prototipo de la interfaz gráfica.

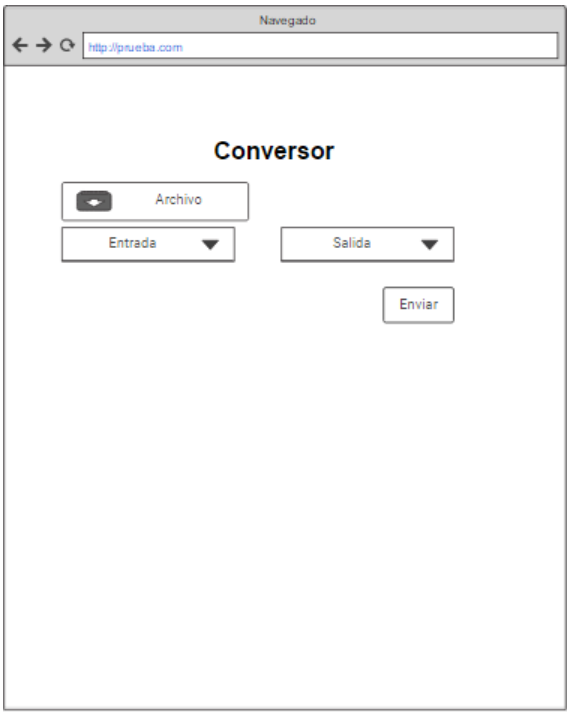

Figura 4.2. Prototipo de la interfaz

5. Implementación.

En esta sección se documenta la fase de implementación del proyecto. En primer lugar veremos algunos aspectos generales del proyecto para después comentar con más detalle las partes más interesantes así como los problemas y las soluciones encontradas.

A lo largo de la fase de implementación, haremos uso de varias tecnologías y de varias librerías de terceros. Las tecnologías que vamos a utilizar aparecen ya mencionadas en la planificación por lo que haremos hincapié en las librerías de terceros empleadas.

En un primer momento se decantó por hacer una clase JAVA, que creara los archivos XML siguiendo el XML esquema de PopXML conforme se leía el documento plano que nos dan, pero el coste de crear una clase con la librería Javax.xml, con su árbol DOM era muy costoso por lo que se buscó una alternativa mejor y más sencilla, que era la de utilizar JAXB para generar las clases que nos permitieran crear el documento XML más rápidamente sin tener que escribir más código.

Librerías:

- JAXB: Esta librería proporciona, la generación de clases Java para la creación de archivos XML a partir de un esquema XML. Al igual que las distintas herramientas para la creación del XML.
- Javax xml transform: Son una serie de APIs XSLTs que permiten escribir los datos XML en un archivo plano aplicando la XSLTs.

Para usar la librería de JAXB, es necesario utilizar un comando para que, de un fichero con la extensión .xsd en el cual se tiene el esquema XML de PopXML, genere clases Java para la creación de un fichero XML con el esquema de PopXML.

Para ello tendremos que descargar la librería jaxb. Una vez descargada tenemos que poner la carpeta el directorio en C, para evitar problemas, seguido abriremos la consola de comandos en el directorio que contenga nuestro .xsd, y ejecutamos el siguiente código como se muestra en la Figura 5.1.

```
C:\Users\David\Desktop\TFG\Pruebas\XML>C:\jaxb-ri-2.2.11\jaxb-ri\bin\xjc.bat
geljXMLModel.xsd -p xml
Analizando un esquema...
Compilando un esquema...
xml\ObjectFactory.java
xml\Study.java

C:\Users\David\Desktop\TFG\Pruebas\XML>
```

Figura 5.1. Ejecución de comando XCJ.

La ejecución del comando es coger *xjc.bat*, aplicarle un .xsd, después *–p* y seguido el nombre de la carpeta que almacenara los ficheros generados, con estos ficheros ya podríamos crear los ficheros XML que seguirán el esquema PoPXML.

Nos hemos decantado por esta solución debido a que el procedimiento de creación de un XML de estas características era muy costoso, por lo que se investigó y se creyó oportuno utilizar JAXB.

Como ya tenemos las clases para poder crear los XML y también se tienen los XSLTs para aplicarlos al XML, se dividirá el proyecto en dos fases:

- La primera es la creación del XML a partir de un documento de texto plano.
- La segunda es el proceso de aplicar al documento XML el XSLTs que nos dará la salida del tipo de herramienta que se elija en la aplicación web.

Ilustración 5.1. Proceso de la aplicación.

La implementación quedará estructurada en incrementos. El fin de cada incremento añadirá más funcionalidad a la aplicación final. De esta forma, mostraremos una evolución en la implementación a lo largo del desarrollo de la misma.

Cada incremento lo dividiremos en tres partes:

- Objetivos: funcionalidad que esperamos realizar durante el incremento.
- Desarrollo: se describirá, el proceso llevado a cabo para implementar las funcionalidades que se dan en los objetivos.
- Cierre: se reflejaran las pruebas realizadas a lo largo de la iteración como los problemas que hemos encontrado y el proceso que hemos seguido para solucionarlos.

5.1. Transformación del fichero plano a XML.

En esta primera fase de la aplicación web, se desarrolló un programa en el cual se utilizaban las clases generadas por el comando XCJ (figura 5.1), para la elaboración de ficheros XML, esto se realiza debido a que se quiere seguir el proceso de creación del XML para comprobar, que la creación de los ficheros XML es la correcta e incluye en el XML toda la información que contiene el fichero plano que se introduce como parámetro de entrada.

Para el programa que nos crea el XML hemos seleccionado el siguiente paquete de trabajo (Figura 5.2):

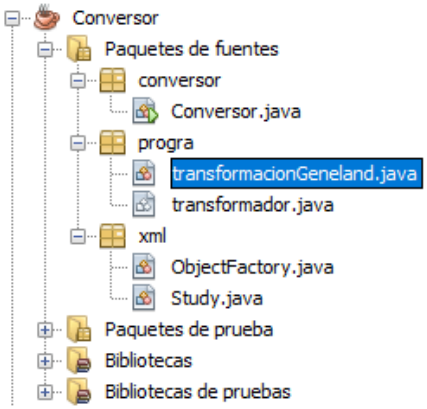

Figura 5.2. Paquete de trabajo.

Se pueden observar en la figura 5.2 que tenemos 3 paquetes:

- *conversor*: Tiene la clase principal para llamar a los métodos del paquete *progra*.
- *progra*: Tiene la clase *transformador* que es abstracta, de la cual heredaran todas las demás clases de este paquete, estas clases utilizan el paquete *xml* para transformar el fichero plano en un fichero XML.
- *xml*: Tiene las clases del comando XCJ que son ObjectFactory y Study con las cuales podemos crear ficheros XML con el esquema PopXML.

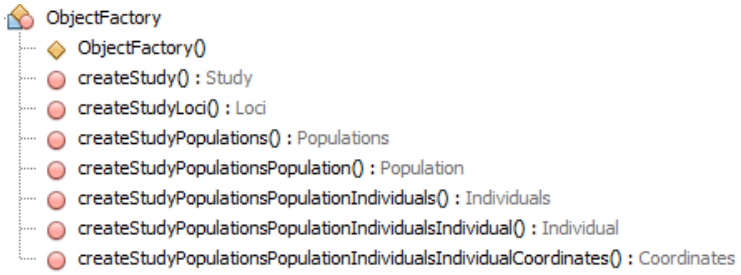

Figura 5.3. Métodos de ObjectFactory.

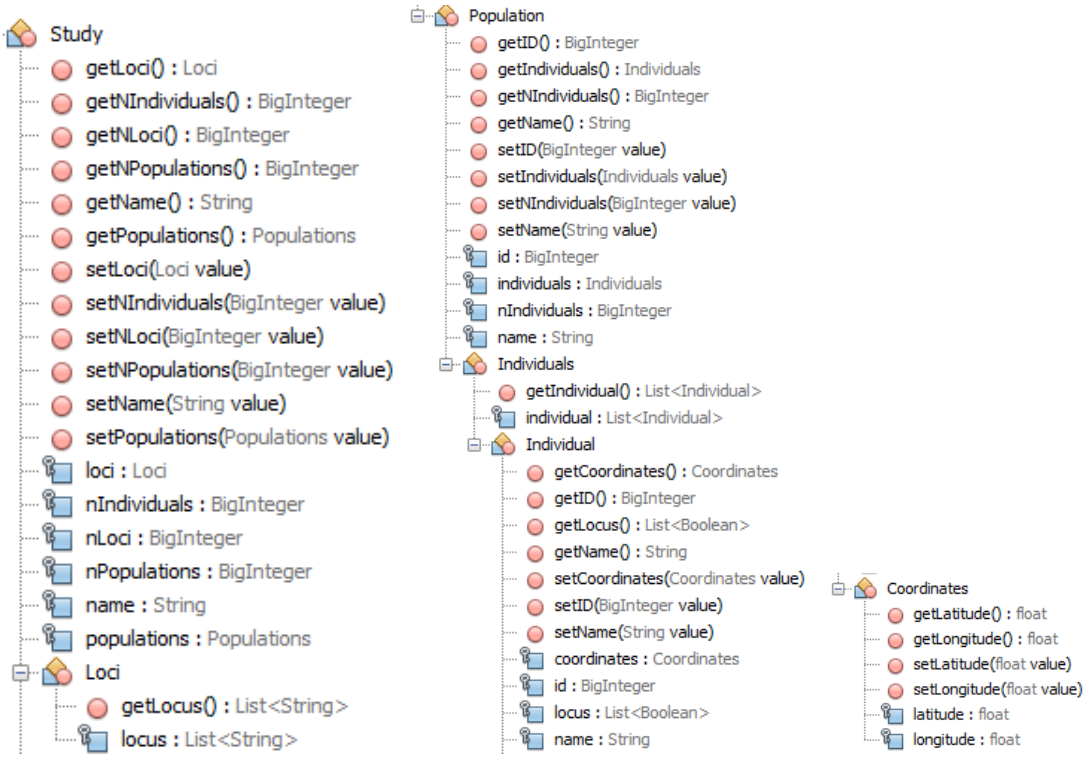

Figura 5.4. Esquema de la clase Study.

Para todas las pruebas que se van a desarrollar se utilizarán los ficheros planos proporcionados por el cliente, estos ficheros planos se transformaran a un XML, que se comparará con el XML intermedio de GelJ con el cual se crearon los ficheros planos.

Para ello hemos seleccionado las prioridades del cliente haciendo los cambios de las 3 herramientas que el creyó oportunas implementar.

5.1.1 1ª Iteración.

Objetivos.

El objetivo de esa iteración es implementar la transformación de un fichero plano de la herramienta de Geneland, a un fichero XML con la estructura del esquema de PoPXML, para que en la segunda fase de la implementación se le pueda aplicar el XSLT.

Desarrollo.

Gracias al fichero XSLT (Figura 5.5) se puede ver como es la entrada de los ficheros de Geneland, para hacer la transformación a XML.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    exclude-result-prefixes="xs"
    version="2.0">
    <xsl:output method="text" omit-xml-declaration="yes" indent="yes"/>
    <xsl:template match="study">
        <!-- Individual lines -->
        <!-- presence/absence of loci using 1/0, id of the individual-->

        <xsl:for-each select="populations/population">
            <xsl:for-each select="individuals/individual">
                <xsl:for-each select="locus">
                    <xsl:if test=". = 'false'">
                        <xsl:text>0</xsl:text>
                    </xsl:if>
                    <xsl:if test=". = 'true'">
                        <xsl:text>1</xsl:text>
                    </xsl:if>
                    <xsl:text> </xsl:text>
                </xsl:for-each>
                <xsl:text>&#xA;</xsl:text>
            </xsl:for-each>
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

Figura 5.5. XSLTs de Geneland.

```
0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 1 0 0 0 1 0 1 1 1 0 0 1 1 0 0 1 0 0 1 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0
0 0 1 1 0 1 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 1 0 1 1 0 0 0 1 1 0 1 1 0 0 1 1 0 0 1 0 1 1 0 1 1 0 0 1 0 0
1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1 0 1 0 1 0 1 0 1 1 0
0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 1 1 0 1 0 1 1 0
0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 1 1 0 1 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 0 1 0 0
0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 1 1 0 1 0 1 1 0 0 1 1 0 0 1 0 1 0 1 0 1 1 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0 0 1 0 0 0 0 1 0 1 1 1 1 1 0 0 1 1 0 0 0 0 0 1 1 1 1 1 1 1 0 1 1 0
0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 1 0 0 0 0 0 1 1 1 1 0 1 0 0 1 1 1
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 0 0 0 0 0 1 1 1 0 1 0 1 0 0 1 1 0
0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0 0 1 0 1 0 0 1 0 0 0 1 1 0 0 0 0 1 1 1 0 1 0 1 0 0 1 1 0
0 0 0 0 0 1 0 1 1 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 1 1 1 0 1 0 1 1 0 1 1 1 0 1 0 0 1 1 0 0 0 1 0 0 1 1 0 0 0 1 0
0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 1 0 0 0 1 1 0 1 0 0 1 1 0 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0 1 1 0 1 1 1 0
```

Figura 5.6. Fichero de Geneland.

Como ya hemos explicado anteriormente nuestro proceso es coger los ficheros de entrada de Geneland como el que se muestra en la figura 5.6 y crear un XML con la información que nos muestra este archivo.

La información que nos muestra un fichero de la herramienta Geneland, gracias a la figura 5.5, son la matriz de presencia/ausencia de los locus, en los cuales si se da false se pone un 0 y si se da true se pone un 1, por lo que hay que crear un XML en el que se muestren una lista de individuales con sus respectivos locus.

El proceso para crear el XML es el mismo en todas las iteraciones, en primer lugar se va leyendo el fichero plano línea por línea, como tenemos el fichero XSLT de cada herramienta, sabemos que información ha de contener el fichero XML, por lo que el fichero lo creamos gracias a las clases generadas con el comando XCJ.

Como se muestra en la figura 5.7 se recorre la línea, y comparando carácter a carácter a ver si coincide con 0 o 1, si se da el caso de 0 se añade false al locus y en caso contrario se añade true.

```
for (int i = 0; i< numeros.length; i++){

    BigInteger bi = new BigInteger(String.valueOf(lineas));
    popu.setNIndividuals(bi);
    if (numeros[i].equals("0")){
        indi.getLocus().add(Boolean.FALSE);
        //System.out.println("entra 0");

    }
    else{
        indi.getLocus().add(Boolean.TRUE);
        //System.out.println("entra 1");
    }
}
```

Figura 5.7. Insertar los Locus de Geneland.

La creación del fichero XML con la entrada de un fichero Geneland se muestra en la siguiente Figura 5.8.

Como se observa en la Figura 5.8 se instancia el contexto con JAXBContext, indicando la clase que será el RootElement, seguidamente se crea un Marshaller, que es la clase capaz de convertir nuestro fichero en una cadena XML. Y la salida se hace mediante un fichero, y también mediante la salida del programa.

```
//fichero de salida
File file = new File("C:\\Users\\David\\Desktop\\TFG\\Pruebas\\Conversor\\Documentos\\salida.xml");
//Proceso para crear el xml
JAXBContext jaxbContext = JAXBContext.newInstance(Study.class);
Marshaller jaxbMarshaller = jaxbContext.createMarshaller();

jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
jaxbMarshaller.marshal(study, file);
jaxbMarshaller.marshal(study, System.out);


}
catch(Exception e){
    e.printStackTrace();
}
//utilizamos el finally para cerrar el fichero
finally{
    try{
        if(null != fr){
            fr.close();
        }
    }
    catch (Exception e2){
        e2.printStackTrace();
    }
}
```

Figura 5.8. Creación del Fichero XML.

Cierre.

Pruebas.

Para la ejecución del programa se ha creado un principal.

```
transformacionGeneland tg = new transformacionGeneland();
File fichero = new File ("C:\\Users\\David\\Desktop\\TFG\\Pruebas\\Conversor\\Documentos\\Test_geneland_data.txt");
tg.convierte(fichero);
```

Figura 5.9. Main de la aplicación XML.

Como se observa en la Figura 5.9 nuestro programa llama a *transformacionGeneland*, posteriormente se le dice cuál es el fichero a transformar en XML y por último se llama al método *convierte* con el fichero a cambiar.

Como se puede observar, se ha cumplido el objetivo de implementar la transformación a XML de los ficheros planos de la herramienta Geneland.

Incidencias.

Como el proceso de lectura del fichero de entrada es mediante un while, se producía un incidente inesperado cuando se cambiaba el orden de creación de algunos componentes del fichero como, las poblaciones y los individuos, por ello como se puede observar en la Figura 5.10, las poblaciones y los individuos se tienen que crear dentro del while y fuera del mismo la población y el individuo.

Esto es debido a que el while es un proceso iterativo que pisaba el proceso anteriormente hecho por lo que la salida originaba un único individuo en lugar de 12.

```
//Crear individuals
Study.Populations.Population.Individuals individuals = new Study.Populations.Population.Individuals();
//Crear population
Study.Populations.Population popu = new Study.Populations.Population();

while ((linea = br.readLine()) != null){

    //Crear populations
    Study.Populations populations = new Study.Populations();
    //Crear individual
    Study.Populations.Population.Individuals.Individual indi = new Study.Populations.Population.Individuals.Individual();
```

Figura 5.10. Separación de los inicializadores.

5.1.2 2ª Iteración.

Objetivos.

El objetivo de esa iteración es implementar la transformación de un fichero plano de la herramienta de Genepop, a un fichero XML con la estructura del esquema de PoPXML, para que en la segunda fase de la implementación se le pueda aplicar el XSLT.

Desarrollo.

Gracias a los ficheros XSLT se puede ver como es la salida de los ficheros planos, para hacer la transformación a XML.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    exclude-result-prefixes="xs"
    version="2.0">
    <xsl:output method="text" omit-xml-declaration="yes" indent="yes"/>
    <xsl:template match="study">
        <!-- First line -->
        <!-- Study name -->
        <xsl:value-of select="name"/>
        <xsl:text>&#xA;</xsl:text>

        <!-- second line -->
        <!-- loci names -->
        <xsl:for-each select="loci/locus">
            <xsl:value-of select="."/>
            <xsl:text>&#xA;</xsl:text>
        </xsl:for-each>
        <!-- Following lines -->
        <!-- population number, individual name, presence/absence codified by 1/2 -->

        <xsl:for-each select="populations/population">
            <xsl:text>Pop</xsl:text>
            <xsl:text>&#xA;</xsl:text>
            <xsl:for-each select="individuals/individual">
                <xsl:value-of select="name"/>
                <xsl:text>,</xsl:text>
                <xsl:for-each select="locus">
                    <xsl:if test=". = 'false'">
                        <xsl:text>2</xsl:text>
                    </xsl:if>
                    <xsl:if test=". = 'true'">
                        <xsl:text>1</xsl:text>
                    </xsl:if>
                    <xsl:if test="not(last()=position())">
                        <xsl:text> </xsl:text>
                    </xsl:if>
                </xsl:for-each>
                <xsl:text>&#xA;</xsl:text>
            </xsl:for-each>
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

Figura 5.11. XSLT de Genepop.

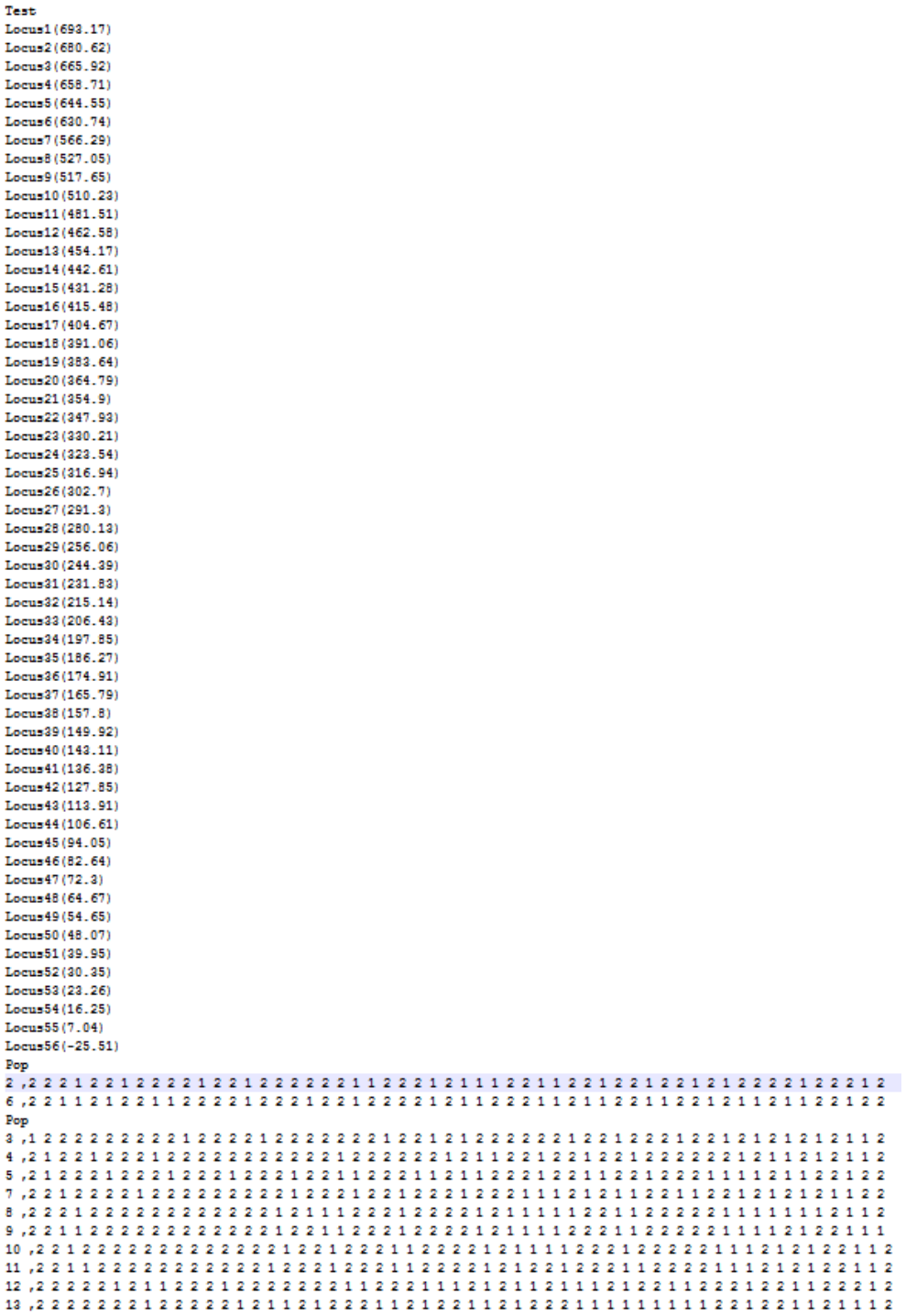

Figura 5.12. Fichero de Genepop.

Aquí vemos por la Figura 5.11 que la información que va a tener los archivos Genepop es mayor que los proporcionados por los ficheros de Geneland.

La información que nos muestra un fichero de la herramienta Genepop, gracias a la figura 5.11, son el nombre del estudio, el nombre de cada locus, y la matriz de presencia/ausencia de los locus de cada población, a diferencia de Geneland que, para almacenar la matriz los divide con la cadena “Pop” para referirse a que las

siguientes líneas pertenecen a la misma población, en la cual si es false se pone un 2 y si es true se pone un 1.

Para la creación del XML se va leyendo el fichero línea por línea, como sabemos que es lo que tiene que contener el XML gracias al fichero XSLT y al ejemplo de entrada, la primera línea siempre es el nombre del estudio, las siguientes líneas son los nombres de los locus, por ello se compara la cadena con "Locus" para ir insertándolos en el XML, seguido hay que comparar la línea con la cadena "Pop", para saber cuándo empiezan las poblaciones y los locus que pertenecen a cada población que hay dentro de dichas poblaciones, después de la cadena "Pop", se leen las siguientes líneas para ir insertando los locus, hasta que haya un nuevo "Pop", que en este caso se repetiría el mismo proceso, o se acabe el fichero.

En la siguiente Figura 5.13 vemos como introducimos los Locus en una lista para después insertarlo en el fichero XML (Figura 5.14).

```
//Ponemos el numero de Locus
if(linea.contains("Locus")){
    numLocus ++;
    study.setNLoci(new BigInteger(String.valueOf(numLocus)));
    listaLocus.add(linea);
}
```

Figura 5.13. Almacenamiento de Locus en la variable.

```
//Insertar locus
Study.Loci loci = new Study.Loci();
for(int i = 0; i < listaLocus.size(); i++){
    loci.getLocus().add(listaLocus.get(i));
}
study.setLoci(loci);
```

Figura 5.14. Inserción de los Locus en Genepop.

En la siguiente Figura 5.15 observamos en primer que almacenamos el número de poblaciones que contiene, y en segundo lugar almacenamos en una lista de listas los locus, para diferenciar todos los locus que hay en el fichero, por ello se lee la cadena "Pop" para saber que a continuación vienen los locus de la población, y que cada línea pertenecerá a cada individuo dicha población.

```
//Ponemos el numero de Populations
if(linea.contains("Pop")){
    numeroPop ++;
    study.setNPopulations(new BigInteger(String.valueOf(numeroPop)));
    afterPop = true;
}

//Ponemos el numero de individuals
if(afterPop == true){
    if(linea.contains("Pop")){
        listalistaIndi.add(new ArrayList<>());
        numlistIndi ++;
    } else{
        numIndi ++;
        study.setNIndividuals(new BigInteger(String.valueOf(numIndi)));
        listalistaIndi.get(numlistIndi).add(linea);
    }
}
```

Figura 5.15. Almacenamiento de los Locus en una variable.

Una vez llenada la lista de listas con los locus de cada individuo se procesa como se muestra en la Figura 5.16 para insertarlo en el fichero XML.

```
for (int y = 0; y < listaindi.size(); y ++){
    //Crear individual
    Study.Populations.Population.Individuals.Individual individual = new Study.Populations.Population.Individuals.Individual();
    //Hacer de la linea que contiene el individual una lista con los numeros del individual
    String cadena = listaindi.get(y);
    String delimitadores = "[ .,;?!¡¿\'\"\\[\\]]+";
    String[] numeros = cadena.split(delimitadores);
    //Incrementamos uno cada vez que entre en un individual para despues asignarselo al ID
    indiID ++;

    for(int x = 1; x < numeros.length ; x ++){
        //Recorremos la lista para insertar el Locus de cada individual
        if (numeros[x].equals("2")){
            individual.getLocus().add(Boolean.FALSE);
        }
        else{
            individual.getLocus().add(Boolean.TRUE);
        }
        //Asignar valor al ID de cada individual
        individual.setID(new BigInteger(String.valueOf(indiID)));
        //Asignar valor al nombre de cada individual
        individual.setName(numeros[0]);
    }
    //Incluimos en cada lista de individuals el individual
    individuals.getIndividual().add(individual);
}
```

Figura 5.16. Inserción de los Population.

Una vez rellenados todos los campos se ejecuta el mismo código que se utilizó para generar el XML de Geneland en la Figura 5.8.

Cierre.

Pruebas.

Para la ejecución del programa se ha creado un principal.

```
transformacionGenepop tgp = new transformacionGenepop();
File ficheropop = new File ("C:\\Users\\David\\Desktop\\TFG\\Pruebas\\Conversor\\Documentos\\Test_genepop.txt");
tgp.convierte(ficheropop);
```

Figura 5.17. Main de la aplicación XML.

Como se observa en la Figura 5.17 nuestro programa llama a *transformacionGenepop*, posteriormente se le dice cuál es el fichero a transformar en XML y por último se llama al método *convierte* con el fichero a cambiar.

El objetivo de la iteración se ha cumplido, puesto que se ha almacenado toda la información en el documento XML.

Incidencias.

Como nos pasó en la anterior iteración, cuando se inicializaban los procesos de creación de los componentes del XML dentro del while, producía que la salida estuviera mal formada, para ello en esta iteración al ser más compleja se decidió, almacenar todos los datos en variables para después del while, poder procesar toda la información sin problemas en la salida.

```
//Numero de lineas que tiene el fichero
int lineas = 0;
//El numero dePopulations que tiene.
int numeroPop = 0;
//El numero de locus
int numLocus = 0;
//Creamos una lista con todos los Locus
List<String> listaLocus = new ArrayList<String>();
//Creamos un booleano para saber cuando tiene que insertar los individuals
boolean afterPop = false;
//Numero de individuals
int numIndi = 0;
//Lista de listas con los individuals;
List<List<String>> listalistaIndi = new ArrayList<List<String>>();
//Recorrer la listalistaIndi
int numlistIndi = -1;
```

Figura 5.18. Variables en Genepop.

5.1.3 3ª Iteración.

Objetivos.

El objetivo de esa iteración es implementar la transformación de un fichero plano de la herramienta de GenAlEx, a un fichero XML con la estructura del esquema de PoPXML, para que en la segunda fase de la implementación se le pueda aplicar el XSLT.

Desarrollo.

Gracias a los ficheros XSL se puede ver como es la salida de los ficheros planos, para hacer la transformación a XML. Como se muestra el Fichero XSL de la herramienta GenAlEx.

```
<xsl:template match="study">
    <!-- First line -->
    <!-- No. Loci,No. Samples,No. Populations,Size of each population -->
    <xsl:value-of select="n_loci"/>
    <xsl:text>,</xsl:text>
    <xsl:value-of select="n_individuals"/>
    <xsl:text>,</xsl:text>
    <xsl:value-of select="n_populations"/>
    <xsl:text>,</xsl:text>
    <xsl:for-each select="populations/population">
        <xsl:value-of select="n_individuals"/>
        <xsl:if test="not(last()=position())">
            <xsl:text>,</xsl:text>
        </xsl:if>
    </xsl:for-each>
    <xsl:text>&#xA;</xsl:text>
    <!-- Second line -->
    <!-- Name study, , names of populations -->
    <xsl:value-of select="name"/>
    <xsl:text>,,</xsl:text>
    <xsl:for-each select="populations/population">
        <xsl:value-of select="name"/>
        <xsl:if test="not(last()=position())">
            <xsl:text>,</xsl:text>
        </xsl:if>
    </xsl:for-each>
    <xsl:text>&#xA;</xsl:text>
    <!-- Third line -->
    <!-- "Sample", "Population", loci names -->
    <xsl:text>Sample</xsl:text>
    <xsl:text>,</xsl:text>
    <xsl:text>Population</xsl:text>
    <xsl:text>,</xsl:text>
    <xsl:for-each select="loci/locus">
        <xsl:value-of select="."/>
        <xsl:if test="not(last()=position())">
            <xsl:text>,</xsl:text>
        </xsl:if>
    </xsl:for-each>
    <xsl:text>&#xA;</xsl:text>
    <!-- Following lines -->
    <!-- Sample name, Population name, loci values -->
    <xsl:for-each select="populations/population">
        <xsl:variable name="popnmame" select="name"/>
        <xsl:for-each select="individuals/individual">
            <xsl:value-of select="name"/>
            <xsl:text>,</xsl:text>
            <xsl:value-of select="$popnmame"/>
            <xsl:text>,</xsl:text>
            <xsl:for-each select="locus">
                <xsl:if test=". = 'false'">
                    <xsl:text>0</xsl:text>
                </xsl:if>
                <xsl:if test=". = 'true'">
                    <xsl:text>1</xsl:text>
                </xsl:if>
                <xsl:if test="not(last()=position())">
                    <xsl:text>,</xsl:text>
                </xsl:if>
            </xsl:for-each>
            <xsl:text>&#xA;</xsl:text>
        </xsl:for-each>
    </xsl:for-each>
</xsl:template>
```

Figura 5.19. XSLT de GenAlEx.

```
56,12,2,2,10
Test,,G1,defpop
Sample,Population,Locus1(693.17),Locus2(680.62),Locus3(665.92),Locus4(658.71),Locus5(644.55),Locus6(630.74),Locus7(566.29),Locus8(527.05),Locus9(517.65),Locus10(51
0.23),Locus11(481.51),Locus12(462.58),Locus13(454.17),Locus14(442.61),Locus15(431.28),Locus16(415.48),Locus17(404.67),Locus18(391.06),Locus19(383.64),Locus20(364.7
9),Locus21(354.9),Locus22(347.93),Locus23(330.21),Locus24(323.54),Locus25(316.94),Locus26(302.7),Locus27(291.3),Locus28(280.13),Locus29(256.06),Locus30(244.39),Loc
us31(231.83),Locus32(215.14),Locus33(206.43),Locus34(197.85),Locus35(186.27),Locus36(174.91),Locus37(165.79),Locus38(157.8),Locus39(149.92),Locus40(143.11),Locus41
(136.38),Locus42(127.85),Locus43(113.91),Locus44(106.61),Locus45(94.05),Locus46(82.64),Locus47(72.3),Locus48(64.67),Locus49(54.65),Locus50(48.07),Locus51(39.95),Lo
cus52(30.35),Locus53(23.26),Locus54(16.25),Locus55(7.04),Locus56(-25.51)
2 ,G1,0,0,0,1,0,0,1,0,0,0,0,1,0,0,1,0,0,0,0,0,0,1,1,0,0,0,1,0,1,1,1,0,0,1,1,0,0,1,0,1,0,1,0,0,0,0,1,0,0,0,1,0
6 ,G1,0,0,1,1,0,1,0,0,1,1,0,0,0,0,1,0,0,0,1,0,0,1,0,0,0,0,1,0,1,1,0,0,0,1,1,0,0,1,1,0,0,1,0,1,1,0,1,1,0,0,1,0,0
3 ,defpop,1,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,1,0,1,0,0,0,0,0,0,1,0,0,1,0,0,1,0,1,0,1,0,1,0,1,1,0
4 ,defpop,0,1,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,1,1,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,1,0,1,1,0,1,0,1,1,0
5 ,defpop,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,1,1,0,0,0,1,1,0,1,1,0,0,0,1,0,0,1,1,0,0,1,0,0,0,1,1,1,1,0,1,1,0,0,1,0,0
7 ,defpop,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,1,0,0,0,1,0,0,0,0,1,0,0,0,1,1,1,0,1,0,1,1,0,0,1,1,0,0,1,0,1,0,1,0,1,1,0,0
8 ,defpop,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,1,1,0,0,0,1,0,0,0,0,0,1,0,1,1,1,1,1,1,0,0,1,1,0,0,0,0,0,1,1,1,1,1,1,1,1,0,1,1,0
9 ,defpop,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,1,0,0,0,1,0,0,0,0,0,1,0,1,1,1,1,1,0,0,0,1,1,0,0,0,0,0,0,1,1,1,1,1,0,1,0,0,1,1,1
10 ,defpop,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,1,0,0,0,0,0,1,0,1,1,1,1,1,0,0,0,1,0,0,0,0,0,0,1,1,1,0,1,0,1,0,0,1,1,0
11 ,defpop,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,1,1,0,0,0,0,0,1,0,1,0,0,1,0,1,0,0,0,0,1,1,0,0,0,0,0,1,1,1,0,1,0,1,0,0,1,1,0
12 ,defpop,0,0,0,0,0,1,0,1,1,0,0,0,1,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,1,0,1,0,1,1,0,1,1,1,0,1,0,0,1,1,0,0,0,1,0,0,1,1,0,0,0,0,1,0
13 ,defpop,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,1,1,0,1,0,0,0,0,1,1,0,1,0,0,1,1,0,1,0,1,0,0,0,1,1,1,1,1,1,1,1,1,0,0,1,0,0,1,1,0,1,1,1,1,0
```

Figura 5.20. Fichero de GenAlEx

Como se puede observar de la Figura 5.19 vemos que la información que ofrece esta herramienta es completa respecto con el esquema XML de PopXML, se puede ver como muestra toda la información, mostrando el número de Locus que tiene el fichero, el número de poblaciones, y también el número de individuos, y el número de individuos que hay en cada población y el nombre de cada población, como todos los loci que contiene y también almacena los individuos junto con su respectiva población. La lista de los locus de los individuos se almacena como false si pone un 0 y true si pone un 1.

El proceso para la creación del XML es igual que respecto a la anterior iteración, pero en esta hay que tener en cuenta que no se nos da un separador de cada población como “Pop”, sino que hay que comparar los nombres de cada población para ver si son iguales, y después insertarlos en dicha población.

En la Figura 5.21 vemos como almacenamos los locus en la variable para después insertarlos en el XML como se muestra en la Figura 5.22.

```
//Poner los loci en la lista para despues procesarlo
if(lineas == 2){
    String cadena = linea;
    String delimitadores = "[ ,;?!¡¿\'\"\\[\\]]+";
    String[] numeros = cadena.split(delimitadores);
    for (int i = 2 ; i < numeros.length ; i ++){
        //System.out.println(numeros[i]);
        if(numeros[i].contains("Locus")){
            listaLocus.add(numeros[i]);
        }
    }
}
```

Figura 5.21. Almacenamiento del Locus en la variable.

```
//Insertar locus
Study.Loci loci = new Study.Loci();
for(int i = 0; i < listaLocus.size(); i++){
    loci.getLocus().add(listaLocus.get(i));
}
```

Figura 5.22. Inserción en el XML los locus.

En la figura 5.23 introducimos todos los individuos en la variable (anteriormente se ha creado una lista para almacenar todos los nombres de las poblaciones para comparar, y saber que individuo va en cada población), para después procesarlo y crear una lista de listas para que se puedan almacenar todos los individuos en su correspondiente población (Figura 5.24) para insertarlos en el XML como se muestra en la Figura 5.25.

```
//Insertar en la lista todos los individuals
for (int x = 0; x < nombrIndis.size(); x++){
    if(lineas != 1){
        if(linea.contains(nombrIndis.get(x))){
            listaIndi.add(linea);
        }
    }
}
```

Figura 5.23. Almacenamiento de los individuos en una variable.

```
//Incluimos en la lista de listas los individuals de cada population.
for (int x = 0; x < nombrIndis.size(); x ++){
    //System.out.println(nombrIndis.get(x));
    listalistaIndi.add(new ArrayList<>());
    for (int i = 0; i < listaIndi.size(); i ++){
        //System.out.println(listaIndi.get(i));
        if (listaIndi.get(i).contains(nombrIndis.get(x))){
            listalistaIndi.get(numlistIndi).add(listaIndi.get(i));
        }
    }
    numlistIndi ++;
}
```

Figura 5.24. Se almacena en la lista de listas los individuos de cada población.

```
for (int y = 0; y < listaindi.size(); y ++){
    //Crear individual
    Study.Populations.Population.Individuals.Individual individual = new Study.Populations.Population.Individuals
    //Hacer de la linea que contiene el individual una lista con los numeros del individual
    String cadena = listaindi.get(y);
    String delimitadores = "[ .,;?!¡¿\'\"\\[\\]]+";
    String[] numeros = cadena.split(delimitadores);
    //Incrementamos uno cada vez que entre en un individual para despues asignarselo al ID
    indiID ++;
    popu.setName(numeros[1]);
    for(int x = 2; x < numeros.length ; x ++){
        //Recorremos la lista para insertar el Locus de cada individual
        if (numeros[x].equals("0")){
            individual.getLocus().add(Boolean.FALSE);
        }
        else{
            individual.getLocus().add(Boolean.TRUE);
        }
        //Asignar valor al ID de cada individual
        individual.setID(new BigInteger(String.valueOf(indiID)));
        //Asignar valor al nombre de cada individual
        individual.setName(numeros[0]);

    }
    //Incluimos en cada lista de individuals el individual
    individuals.getIndividual().add(individual);
}
```

Figura 5.25. Inserción de los individuos de cada población en el XML.

Una vez rellenados todos los campos se ejecuta el mismo código que se utilizó para generar el XML de Geneland en la Figura 5.8.

El objetivo de la iteración se ha cumplido, puesto que se ha almacenado toda la información en el documento XML.

Cierre.

Pruebas.

Para la ejecución del programa se ha creado un principal.

```
transformacionGenAlEx tga = new transformacionGenAlEx();
File ficherogenAl = new File ("C:\\Users\\David\\Desktop\\TFG\\Pruebas\\Conversor\\Documentos\\Test_genAlEx.txt");
tga.convierte(ficherogenAl);
```

Figura 5.26. Main de la aplicación XML.

Como se observa en la Figura 5.26 nuestro programa llama a *transformacionGenAlEx*, posteriormente se le dice cuál es el fichero a transformar en XML y por último se llama al método *convierte* con el fichero a cambiar.

Incidencias.

Hemos aplicado la misma dinámica que en la anterior en la que creamos las variables necesarias para almacenar la información del fichero plano.

```
//Numero de lineas que tiene el fichero
int lineas = 0;
//El numero dePopulations que tiene.
int numeroPop = 0;
//El numero de locus
int numLoci = 0;
//Creamos una lista con todos los Locus
List<String> listaLocus = new ArrayList<String>();
//Creamos un booleano para saber cuando tiene que insertar los individuals
boolean afterPop = false;
//Numero de individuals
int numIndi = 0;
//Lista de individuals
List<String> listaIndi = new ArrayList<String>();
//Lista de listas con los individuals;
List<List<String>> listalistaIndi = new ArrayList<List<String>>();
//Recorrer la listalistaIndi
int numlistIndi = 0;
//Lista de los nombres de los individuales
List<String> nombrIndis = new ArrayList<String>();
//Lista de los numeros de individual de cada population
List<Integer> listanumIndi = new ArrayList<Integer>();
```

Figura 5.27. Variables GenAlEx.

Una vez completado esta fase para ver que funcionaba la generación del XML lo aplicamos a la aplicación web.

5.2. Creación de la aplicación.

Como se comentó anteriormente en este documento la aplicación a desarrollar sería web, por lo que nuestra aplicación tendrá una página de inicio en la cual este el formulario para obtener los datos necesarios para la conversión de archivos, que son la entrada del fichero, un desplegable con las opciones de las herramientas de entrada y un desplegable con las opciones de las herramientas de salida. Como hemos comentado anteriormente la fase 5.1 de creación de XML se integrara en esta fase, para el correcto funcionamiento de la aplicación, que se explicara más adelante. Esto es debido a que es necesaria una transformación a XML para después transformarlo a un fichero plano.

Para la creación de la aplicación se ha utilizado el Patrón de diseño Factory. Debido a que este patrón nos facilita el crear más transformadores de una manera rápida y sencilla, puesto que vamos a tener varias clases que van a tener los mismos métodos. Además de que este patrón nos ayuda a la hora de crear el tipo de objeto que se va a instanciar, puesto que desde un principio no se sabe cuál es la transformación que va a realizar el cliente.

El paquete de trabajo quedaría de la siguiente forma en la figura 5.28.

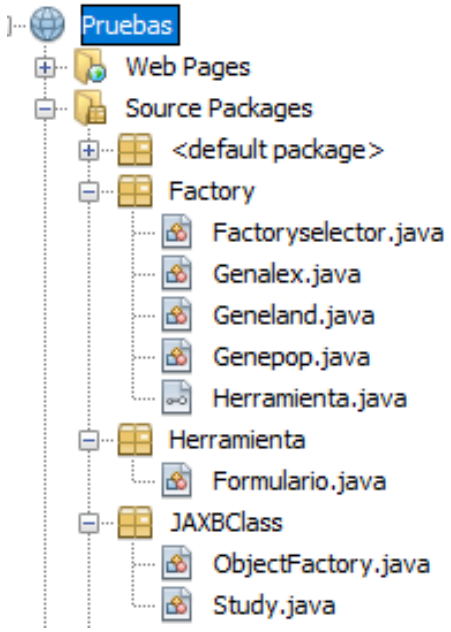

Figura 5.28. Paquete de trabajo de la aplicación.

En la Figura 5.28 se muestra el paquete de trabajo, el cual se divide entre 4 paquetes:

- **Web Pages:** Contiene la página en la cual tomaremos los parámetros de la aplicación.
- **Default package**: Es en el que hemos alojado el esquema XML.
- **Factory**: Es el paquete en el cual se han desarrollado las clases java para el patrón Factory.

- **JAXBClass**: Son las clases para poder generar los archivos XML.
- **Herramienta**: Es la clase para poder recoger la información del formulario.

Ya teniendo claro cuál es la distribución del paquete de trabajo, hay que crear el index de nuestra aplicación en la cual es necesario que el form que tenga la página sea multipart/form-data para poder obtener ficheros del mismo. Este index se apoyará sobre un fichero CSS para hacer la página más cómoda para el cliente. El resultado del mismo se muestra en la ilustración 5.1.

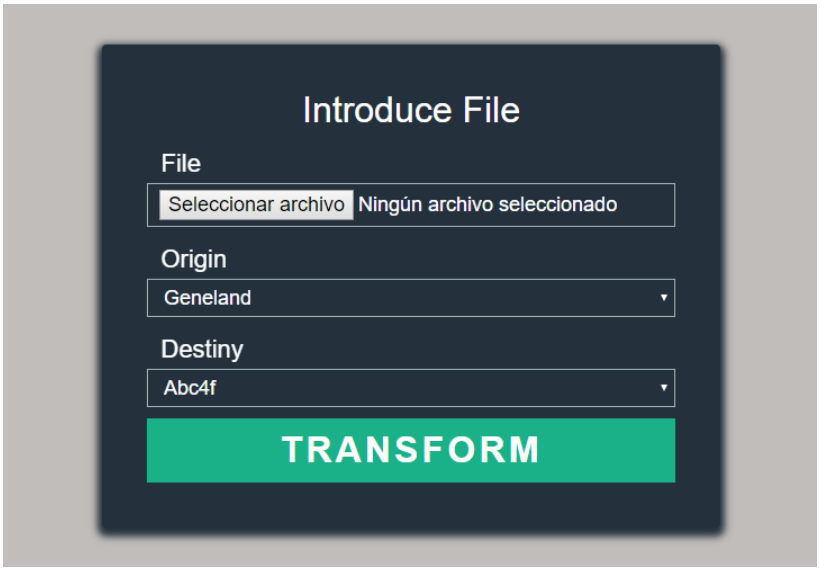

Figura 5.29. Resultado del Index.

Una vez creado el index tenemos que hacer el Servlet para poder procesar los datos que nos da el cliente en la aplicación, en el Servlet (Formulario.java) comprobamos que las variables que introduce el cliente se almacenan correctamente como se muestra en la Figura 5.30.

```
// Se recorren todos los items, que son de tipo FileItem, para leer los parametros del Form
for (FileItem item : items) {
  //Lectura de los parametros
  String campoN = item.getFieldName();
   if (item.isFormField()) { //Si es un parametro normal
      if(campoN.equals("origen")){
          System.out.println("Entra en origen");
          origen = item.getString("UTF-8");
      }
      else if (campoN.equals("destino")){
          System.out.println("Entra en destino");
          destino = item.getString("UTF-8");
      }
   //Comprobamos el fichero que no sea vacio y que tenga nombre
   } else if (campoN.equals("fich") && item.getName() != null && item.getName().trim().length() > 0){
          System.out.println("entra aqui");

        //Creamos el path para ver donde guardamos el archivo que nos deja en el formulario.
        fichName = item.getName();
        String root = getServletContext().getRealPath("/");
        File path = new File(root + "/temp");
        if (!path.exists()) {
            boolean status = path.mkdirs();
        }

        fich = new File(path + "/" + fichName);
        System.out.println(fich.getAbsolutePath());
        item.write(fich);

    }
}
if (fichero!= null){
    System.out.println("Fichero bien entregado");
}
```

Figura 5.30. Comprobación de los parámetros del Form.

Cada herramienta de transformación a implementar va a tener dos métodos como se muestra en la figura 5.31:

- *XML():* crea el XML a partir del fichero plano.
- *XSL():* que aplica el fichero XSLT al fichero XML anteriormente creado, para dar una salida de un fichero plano.

Esto también nos sirve como constructor para las clases que implementaran esta interfaz. Esto se utilizara más adelante.

```
package Factory;

import java.io.File;

/**
 *
 * @author David
 */
public interface herramienta {
    public File XML();
    public File XSL();
}
```

Figura 5.31. Interface herramienta.

```
public static herramienta getHerramientaEntrada(String entrada, File fichero){

    if(entrada.equals("genalex")){
        return new genalex(fichero,null);
    }
    else if (entrada.equals("geneland")){
        return new geneland(fichero,null);
    }
    else if(entrada.equals("genepop")){
        return new genepop(fichero, null);
    }
    return null;
}
```

Figura 5.32. Método para devolver la herramienta de entrada.

```
public static Herramienta getHerramientaSalida(String salida, File fichero){

    if(salida.equals("genalex")){
        return new Genalex(null,fichero);
    }
    else if (salida.equals("geneland")){
        return new Geneland(null,fichero);
    }
    else if(salida.equals("genepop")){
        return new Genepop(null, fichero);
    }
    else if(salida.equals("abc4f")){
        return new Abc4F(null, fichero);
    }
    else if(salida.equals("adegenet")){
        return new Adegenet(null, fichero);
    }
    else if(salida.equals("aflpdat")){
        return new Aflpdat(null, fichero);
    }
    else if(salida.equals("allelesInSpace_coordinates")){
        return new AllelesInSpace_coordinates(null, fichero);
    }
```

Figura 5.33. Método para devolver la herramienta de salida.

Como hemos comentado al principio de esta fase, se iban a incluir los métodos hechos en la fase 5.1 de creación de XML. Por ello en cada transformación de herramienta creada, los métodos *XML()* son los correspondientes a los apartados anteriores de 5.1.1, 5.1.2 y 5.1.3. Nuestra primera parte de la implementación se adecua a este método. Por los que solo tendrán implementado este método las transformaciones anteriormente descritas: Geneland, Genepop y GenAlEx.

5.2.1 Transformación XML a fichero plano.

Como el cliente quiere que se implementen todas las transformaciones a las herramientas que nos ha dado, se explicará cómo se hace en una de las herramientas puesto que las demás implementaciones son iguales, a excepción de cambiar la ruta en la cual se encuentra el XSLT de la herramienta a utilizar.

Vamos a proceder a implantar en la aplicación la transformación de XML a un fichero plano de la herramienta de Geneland. El método a utilizar es *XSL()* , lo que hace es

coger el fichero XML que ha creado el método *XML()* de la herramienta escogida y aplicarle el fichero XSLT para que la salida muestre el fichero plano generado.

```
@Override
public File XSL() {
    TransformerFactory factory;
    factory = TransformerFactory.newInstance();
    BufferedWriter bw;
    File archivo = null;

    //Inicialización del documento de xslt

    //Source xsl = new StreamSource("C:\\Users\\David\\Desktop\\TFG\\PopXML\\PopXML\\XSLTs\\adegenet.xsl");
    try {
        //Aplicacion de la hoja de estilo.
        Transformer transformer = factory.newTransformer(new StreamSource("C:\\Users\\David\\Desktop\\TFG\\PopXML\\PopXML\\XSLTs\\genela
        //Ruta en la cual se va a guardar.
        String ruta = "C:\\Users\\David\\Downloads\\descarga.txt";
        archivo = new File(ruta);
        bw = new BufferedWriter(new FileWriter(ruta));
        //Aplicar la hoja de estilo al fichero XML.
        transformer.transform(new StreamSource(XML), new StreamResult(new FileOutputStream(archivo)));
    } catch (Exception ex) {
        ex.printStackTrace();
    }
    return archivo;
}
```

Figura 5.34. Aplicación de XSLT al fichero XML en Geneland.

Este proceso es el específico para la herramienta Geneland, pero para que se pueda utilizar en cualquiera de las herramientas que el cliente quiera, solo es necesario cambiar en el inicializador de *Transformer* la ruta en la cual está el fichero XSLT de la herramienta a utilizar.

Por ello hay que implementar el método *XSL()* en todas las herramientas creadas, por lo que al final en nuestro paquete *Factory* tendremos un fichero Java por cada herramientas de análisis genético de poblaciones, como se muestra en la Figura 5.35. Cada uno con su método *XSL()* y su ruta correspondiente.

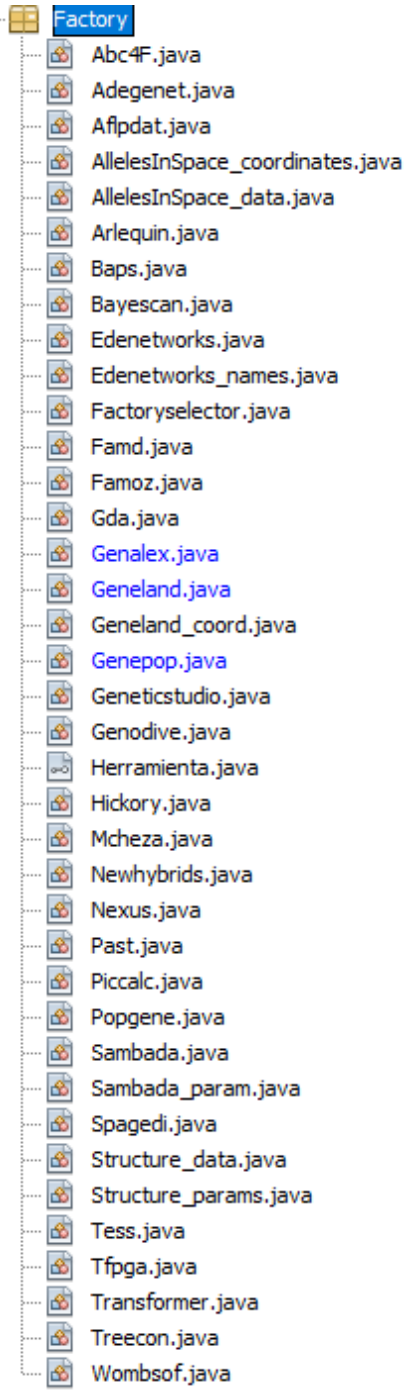

Figura 5.35. Paquete Factory.

5.2.2 Procedimiento de la aplicación.

Ahora procedemos a la creación del XML y aplicación del fichero XSLT dentro de la aplicación web.

Todo este proceso se va a producir dentro del servlet anteriormente creado, por lo que para ello llamamos a los métodos de Factoryselector, que son los anteriormente vistos *getHerramientaEntrada* y *getHerramientaSalida*, así podemos aplicarle los métodos correctos, puesto que a la herramienta que se nos da de entrada se la

transforma a un XML y a la herramienta que nos da de salida se le aplica el XSLT de esa herramienta al XML creado como se ve en la Figura 5.36.

```
Herramienta herraSal = fs.getHerramientaEntrada(origen, fich);
File XML = herraSal.XML();

Herramienta herraEnt = fs.getHerramientaSalida(destino, XML);
salida = herraEnt.XSL();

//Proceso para la salida del fichero
response.setContentType("application/txt");
response.setHeader("Cache-Control", "no-cache"); // HTTP 1.1
response.setHeader("Cache-Control", "max-age=0");
response.setHeader("Content-disposition","attachment; filename=fichero.txt");
ServletOutputStream stream = response.getOutputStream();

FileInputStream input = new FileInputStream(salida);
BufferedInputStream buf = new BufferedInputStream(input);
int readBytes = 0;

while ((readBytes = buf.read()) != -1) {
    stream.write(readBytes);
}

stream.flush();
stream.close();
//Finaliza el proceso de la salida del fichero
```

Figura 5.36. Proceso de transformación y posteriormente salida.

Como podemos observar de la anterior figura, el proceso de salida hay que hacerlo en el método *doPost* de nuestra aplicación por ello se utiliza el método response, en el cual decimos, el tipo de la salida que será, y como se llamará el fichero cuando este se descargue.

Para la salida se utiliza el *ServletOutputStream* que esta enlazado con el método del response que da la salida *getOutputStream* por ello el stream se tiene que llenar leyendo el fichero línea por línea.

6. Pruebas.

El grueso de las pruebas lo hemos llevamos a cabo durante la fase de implementación, reflejando en cada una de las iteraciones e incrementos las pruebas más destacables. En este apartado nos centraremos en la fase final de la aplicación.

Para la realización de las pruebas se hará en dos fases:

- Pruebas en la aplicación en local.
- Pruebas en un servidor.

6.1. Pruebas de la aplicación en local.

En esta fase de las pruebas comprobaremos que las salidas de las distintas pruebas con los ficheros proporcionados por el cliente, son las correctas y en los ficheros que devuelve la aplicación web son los correspondientes.

Para realizar las pruebas se han omitido los ficheros de estilo CSS para que la imagen sea pequeña y que no ocupe mucho espacio, por lo que las figuras que aparecerán no estarán acorde con la Figura 5.31, que nos mostraba el diseño final de la aplicación.

Las pruebas a realizar, serán 3, debido a que hay muchas posibilidades para hacer las transformaciones y tener tantos tipos de herramientas, por ello se ha decantado por hacer las siguientes transformaciones:

- GenAlEx a Geneland.
- GenAlEx a Genepop.
- Genepop a GenAlEx.

Estas 3 pruebas están relacionadas con las 3 herramientas que el cliente dio prioridad, para que además de ver que el resultado es correcto, se pueden comparar con los ficheros que se utilizaron para la creación del XML en el apartado 5.1.

En cada prueba se verán dos figuras, la primera será de la aplicación, en la cual los campos están rellenados con los parámetros que se adecuan a cada prueba, y en la segunda figura será la salida que ofrece nuestra aplicación.

6.1.1 GenAlEx a Geneland

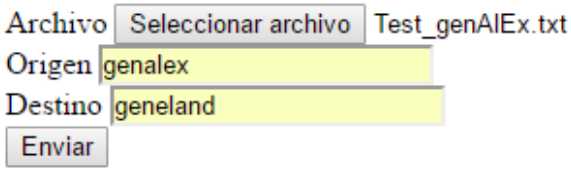

Figura 6.1. Prueba GenAlEx a Geneland en el formulario.

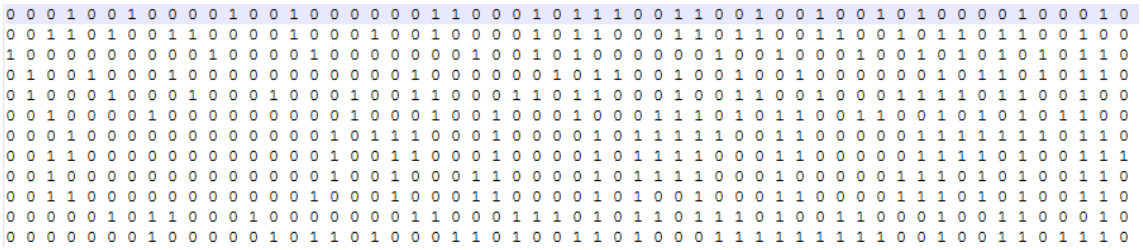

Figura 6.2. Prueba GenAlEx a Geneland salida.

6.1.2 GenAlEx a Genepop

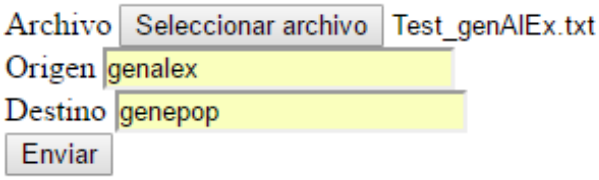

Figura 6.3. Prueba GenAlEx a Genepop formulario.

```
Test
Locus1(693.17)
Locus2(680.62)
Locus3(665.92)
Locus4(658.71)
Locus5(644.55)
Locus6(630.74)
Locus7(566.29)
Locus8(527.05)
Locus9(517.65)
Locus10(510.23)
Locus11(481.51)
Locus12(462.58)
Locus13(454.17)
Locus14(442.61)
Locus15(431.28)
Locus16(415.48)
Locus17(404.67)
Locus18(391.06)
Locus19(383.64)
Locus20(364.79)
Locus21(354.9)
Locus22(347.93)
Locus23(330.21)
Locus24(323.54)
Locus25(316.94)
Locus26(302.7)
Locus27(291.3)
Locus28(280.13)
Locus29(256.06)
Locus30(244.39)
Locus31(231.83)
Locus32(215.14)
Locus33(206.43)
Locus34(197.85)
Locus35(186.27)
Locus36(174.91)
Locus37(165.79)
Locus38(157.8)
Locus39(149.92)
Locus40(143.11)
Locus41(136.38)
Locus42(127.85)
Locus43(113.91)
Locus44(106.61)
Locus45(94.05)
Locus46(82.64)
Locus47(72.3)
Locus48(64.67)
Locus49(54.65)
Locus50(48.07)
Locus51(39.95)
Locus52(30.35)
Locus53(23.26)
Locus54(16.25)
Locus55(7.04)
Locus56(-25.51)
Pop
2,2 2 2 1 2 2 1 2 2 2 2 2 1 2 2 1 2 2 2 2 2 2 2 1 1 2 2 2 1 2 1 1 1 2 2 1 1 2 2 1 2 2 1 2 2 1 2 2 1 2 2 2 2 2 1 2 2 2 2 1 2
6,2 2 1 1 2 1 2 2 1 1 2 2 2 2 1 2 2 2 1 2 2 1 2 2 1 2 2 2 2 2 1 2 1 1 2 2 2 1 1 2 1 1 2 2 1 1 2 2 1 2 1 1 2 1 1 2 2 1 2 2
Pop
3,1 2 2 2 2 2 2 2 2 2 1 2 2 2 2 1 2 2 2 2 2 2 2 1 2 2 1 2 1 2 2 2 2 2 2 1 2 2 1 2 2 2 1 2 2 1 2 1 2 1 2 1 2 1 1 2
4,2 1 2 2 1 2 2 2 1 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 1 2 1 1 2 2 1 2 2 1 2 2 1 2 2 2 2 2 2 1 2 1 1 2 1 2 1 1 2
5,2 1 2 2 2 1 2 2 2 1 2 2 2 1 2 2 2 1 2 2 1 1 2 2 2 1 1 2 1 1 2 2 2 1 2 2 1 2 2 1 1 2 2 1 2 2 2 1 1 1 1 2 1 1 2 2 1 2 2
7,2 2 1 2 2 2 2 1 2 2 2 2 2 2 2 2 2 1 2 2 2 1 2 2 1 2 2 2 1 2 2 2 1 1 1 2 1 2 1 1 2 2 1 1 2 2 1 2 1 2 1 2 1 1 2 2
8,2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 1 2 2 2 1 2 2 2 2 1 2 1 1 1 1 1 2 2 1 1 2 2 2 2 2 2 1 1 1 1 1 1 1 2 1 1 2
9,2 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 1 1 2 2 2 1 2 2 2 2 1 2 1 1 1 1 2 2 2 1 1 2 2 2 2 2 2 1 1 1 1 2 1 2 2 1 1 1
10,2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 1 2 2 2 1 1 2 2 2 2 2 1 2 1 1 1 1 2 2 2 1 2 2 2 2 2 2 1 1 1 2 1 2 1 2 2 1 1 2
11,2 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 1 2 2 2 1 1 2 2 2 2 2 1 2 1 2 2 1 2 2 2 1 1 2 2 2 2 2 1 1 1 2 1 2 1 2 2 1 1 2
12,2 2 2 2 2 1 2 1 1 2 2 2 2 1 2 2 2 2 2 2 2 2 1 1 2 2 2 2 1 1 1 2 1 2 1 1 2 1 1 2 1 2 2 1 1 2 2 2 2 1 2 2 2 1 1 2 2 2 1 2
13,2 2 2 2 2 2 2 1 2 2 2 2 2 2 1 2 1 1 2 1 2 2 2 2 1 1 2 1 2 2 2 1 1 2 1 2 2 2 1 1 2 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1 2 2 1 2 2 1 1 2 1 1 1 2
```

Figura 6.4. Prueba GenAlEx a Genepop salida.

6.1.3 Genepop a GenAlEx

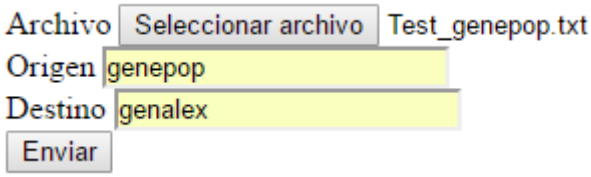

Figura 6.5. Prueba Genepop a GenAlEx formulario.

```
56,12,2,2,10
Test,,,
Sample,Population,Locus1(693.17),Locus2(680.62),Locus3(665.92),Locus4(658.71),Locus5(644.55),Locus6(630.74),Locus7(566.29),Locus8(527.05),Locus9(517.65)
,Locus10(510.23),Locus11(481.51),Locus12(462.58),Locus13(454.17),Locus14(442.61),Locus15(431.28),Locus16(415.48),Locus17(404.67),Locus18(391.06),Locus19
(383.64),Locus20(364.79),Locus21(354.9),Locus22(347.93),Locus23(330.21),Locus24(323.54),Locus25(316.94),Locus26(302.7),Locus27(291.3),Locus28(280.13),Lo
cus29(256.06),Locus30(244.39),Locus31(231.83),Locus32(215.14),Locus33(206.43),Locus34(197.85),Locus35(186.27),Locus36(174.91),Locus37(165.79),Locus38(15
7.8),Locus39(149.92),Locus40(143.11),Locus41(136.38),Locus42(127.85),Locus43(113.91),Locus44(106.61),Locus45(94.05),Locus46(82.64),Locus47(72.3),Locus48
(64.67),Locus49(54.65),Locus50(48.07),Locus51(39.95),Locus52(30.35),Locus53(23.26),Locus54(16.25),Locus55(7.04),Locus56(-25.51)
2,,0,0,0,1,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,1,1,0,0,0,1,0,1,1,1,1,0,0,1,1,0,0,1,0,0,1,0,0,0,1,0,0,1,0,0,0,1,0
6,,0,0,1,1,0,1,0,0,1,1,0,0,0,0,1,0,0,0,1,0,0,1,0,0,1,0,0,0,0,1,0,1,1,0,0,0,1,1,0,1,1,0,0,1,1,0,0,1,0,1,1,0,1,0,0
3,,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,1,0,1,0,0,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,1,0,1,1,0
4,,0,1,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,1,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,1,0,1,1,0,1,0,1,1,0
5,,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,1,1,0,0,0,1,1,0,0,1,1,0,0,0,1,0,0,1,1,0,0,1,0,0,0,1,1,1,1,1,0,1,1,0,0,1,0,1,0,0
7,,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,1,1,0,1,0,1,1,0,0,1,1,0,0,1,0,1,0,1,0,1,1,0,0
8,,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,1,1,1,0,0,0,1,0,0,0,0,0,1,0,1,1,1,1,1,1,0,0,1,1,0,0,0,0,0,0,1,1,1,1,1,1,1,0,1,1,0
9,,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,1,1,0,0,0,1,0,0,0,0,0,1,0,1,1,1,1,1,1,0,0,0,1,1,0,0,0,0,0,0,1,1,1,1,1,0,1,0,0,1,1,1
10,,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,1,1,0,0,0,0,1,0,1,1,1,1,1,0,0,0,1,0,0,0,0,0,0,1,1,1,0,1,0,1,0,1,0,0,1,1,0
11,,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,1,1,0,0,0,0,1,0,1,0,0,0,1,0,0,0,1,1,0,0,0,0,1,1,1,0,1,0,1,0,0,1,1,0
12,,0,0,0,0,0,1,0,1,1,0,0,0,1,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,1,0,1,0,1,1,0,1,1,1,0,1,0,0,1,0,0,1,1,1,0,0,0,1,1,0,0,0,1,0
13,,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,1,1,0,1,0,0,0,1,1,0,1,1,0,1,0,1,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,1,0,1,0,0,1,1,0,1,1,1,0
```

Figura 6.6. Prueba Genepop a GenAlEx salida.

Como se puede observar en las 3 pruebas, las salidas que genera la aplicación son idénticas a los ficheros de entrada de las herramientas de análisis genético de poblaciones además no se ha detectado ningún fallo en la aplicación a la hora de hacer las transformaciones.

6.2. Pruebas en el servidor.

En esta fase de las pruebas vamos a lanzar nuestra aplicación a un servidor, para comprobar el correcto funcionamiento de la aplicación en un servidor. El servidor que se ha elegido es Jelastic, para reducir costes se ha elegido la opción de prueba gratuita, que nos da 15 días de servicio íntegramente.

Para el lanzamiento de la aplicación es necesario generar el fichero .war de nuestra aplicación en el IDE de Netbeans. Una vez generado el fichero .war de la aplicación hay que subirlo dentro de nuestro servidor. Desde la opción de cargar que nos muestra en el panel de control.

Cuando tenemos subido a nuestro servidor el fichero .war de nuestra aplicación es necesario crear un entorno para el lanzamiento de la aplicación, una vez creado todo esto el panel de control quedaría como se muestra en la figura 6.7.

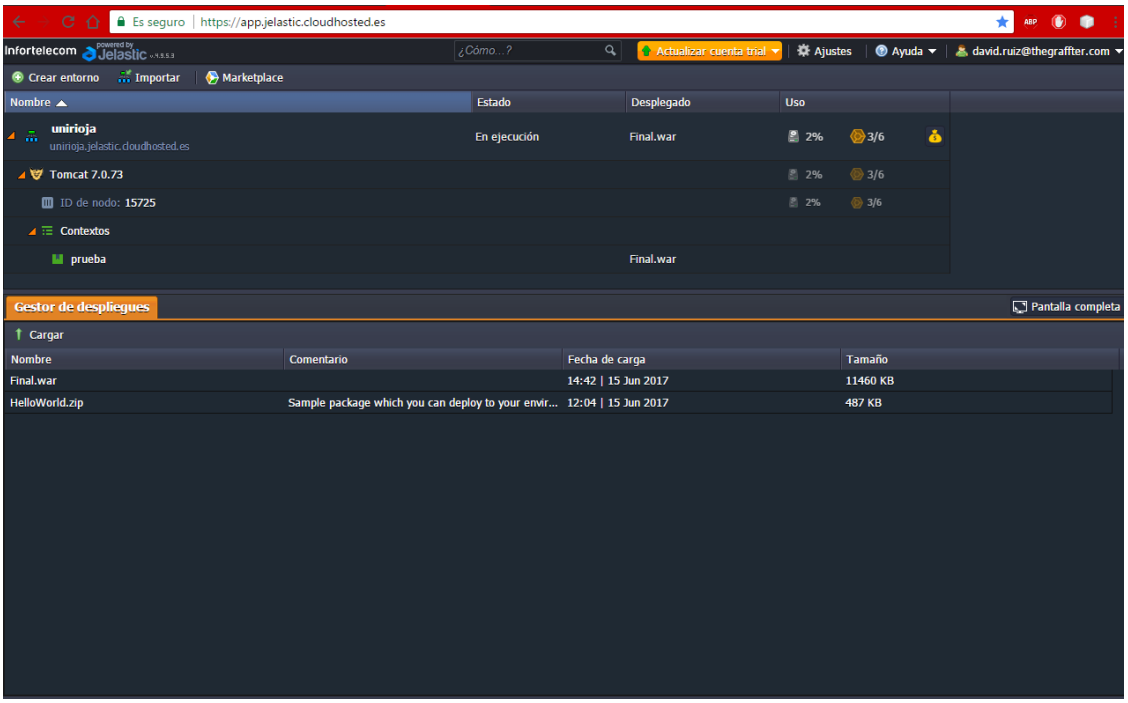

Figura 6.7. Panel de Jelastic.

Una vez subido el fichero .war e inicializado el servidor se elige dentro del servidor ver en navegador como se muestra en la figura 6.8.

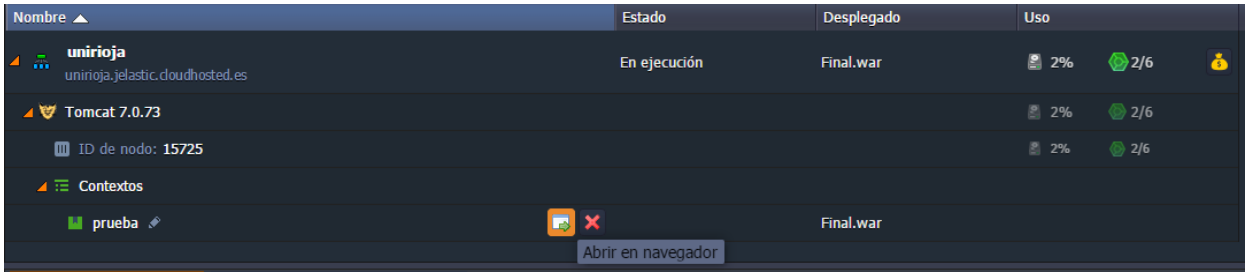

Figura 6.8. Lanzamiento de la aplicación.

El resultado de la aplicación se muestra en la figura 6.9.

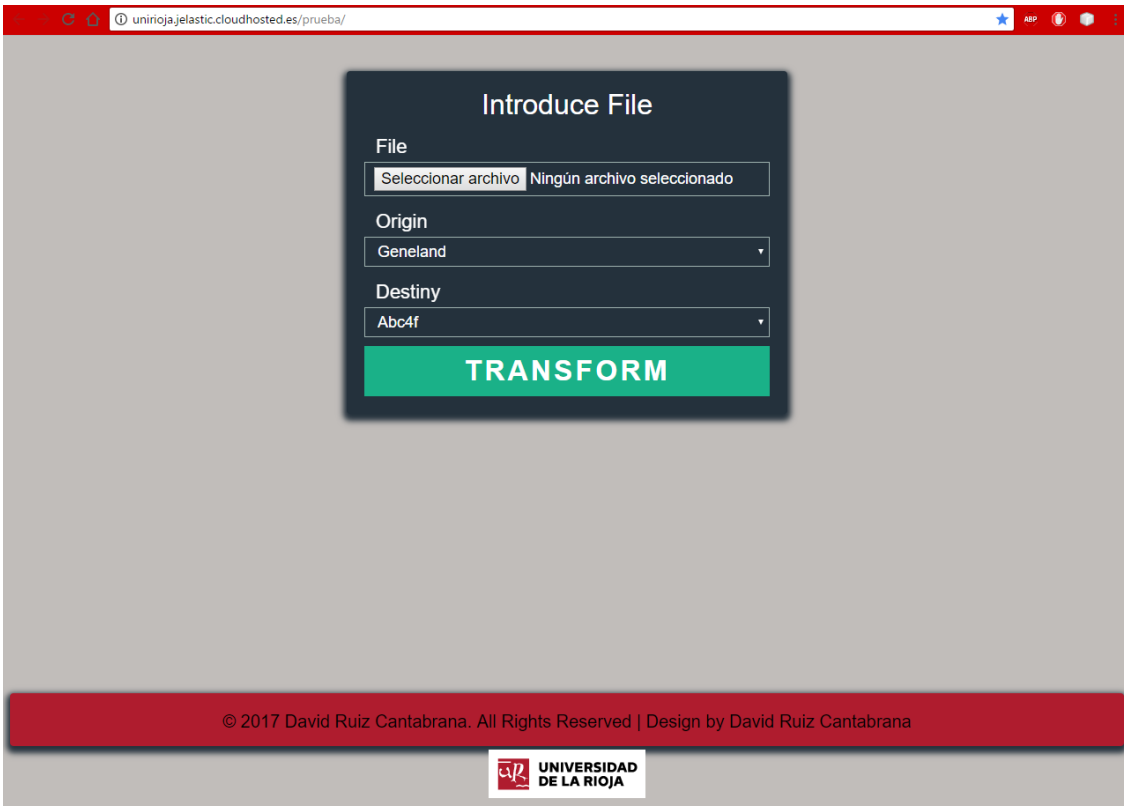

Figura 6.10. Resultado final de la aplicación.

Una vez desplegada, se ha llevado a cabo las mismas pruebas que en la fase anterior y el resultado es el mismo, la aplicación funciona correctamente.

7. Seguimiento y control.

7.1. Plan de seguimiento y control.

A la hora de establecer un plan de seguimiento y control tuvimos en cuenta que hay tres intermediarios dentro de la elaboración del proyecto: tutor académico, cliente y alumno del TFG.

Para que haya un buen seguimiento del trabajo, por parte del cliente y del director, se utilizará un sistema del plan de comunicación para transmitir el avance del mismo:

- Correo electrónico: se utilizará para mandar pequeños documentos, ya sean de la memoria para ver su estructura y su correcta ortografía como del trabajo para ver si la estructura del proyecto es correcta, o se debe modificar algo.
- Reuniones: se utilizará para ver el avance del trabajo y si es procedente cambiar algunas de las partes del mismo, pudiendo hablar cara a cara tanto con el cliente como con el director.

7.2. Seguimiento y Control.

Tras la realización del proyecto se han alcanzado todos los objetivos principales que se identificaron en la planificación, sin realizar ningún cambio en el alcance del mismo. Se ha conseguido hacer una aplicación Web para solucionar el problema de la interoperabilidad entre las herramientas de análisis de poblaciones y que cumpla todos los requisitos identificados en las diferentes reuniones con el cliente.

En la siguiente tabla se muestran las diferentes desviaciones en las tareas a lo largo del proyecto.

Tareas	Horas planificadas	Horas reales	Desviación
T1.1.1 – Planificación.	3H	5H	+2H
T1.1.2 – Planificación de Riesgos.	2H	2,5H	+0.5H
T1.1.3 – Estudio de viabilidad.	3H	4H	+1H
T1.1.4 – Seguimiento y control.	30H	30H	0H
T1.2 – Reuniones.	12H	15H	+3H
T2.1.1 – XSLT.	5H	5H	0H

T2.1.2 – XML schema.	5H	3H	-2H
T2.2.1 – XSLT.	15H	15H	0H
T2.2.2 – XML schema.	10H	15H	+5H
T3.1– Análisis de Requisitos.	15H	12H	-3H
T3.2 – Diseño.	35H	30H	-5H
T3.3 – Implementación.	135H	152H	+17H
T3.4 – Testeo.	30H	30H	0H
Total	300H	318h,5H	+18.5H

Tabla 8.1. Desviaciones.

Como se puede observar en la tabla anterior la mayor desviación ha sido en la fase de implementación, debido a que al principio del proyecto, he tenido que cambiar de método de creación del XML, debido a que el que se había pensado desde un principio era muy costoso y llevaba mucho tiempo hacerlo.

En la fase de formación hay un desvío también debido a que tuve que cambiar de forma de crear los archivos XML y por ello otras librerías y cambiar un poco la estructura de cómo hacerlo.

En la fase de diseño se hizo una estimación un poco mayor de las horas necesarias para realizarlo.

8. Conclusiones.

El grado de satisfacción final de la aplicación es muy alto, se han cumplido los objetivos marcados en el proyecto y se ha entregado al cliente una aplicación funcional que le permite convertir ciertos formatos en otros ofrecidos por la aplicación web. Se han cumplido todos los plazos marcados y los hitos más relevantes del proyecto.

Las pruebas que se han llevado a cabo han resultado óptimas, ajustándose al resultado esperado por la aplicación. La falta de tiempo ha hecho que únicamente se hayan podido hacer pruebas con los ficheros aportados por el cliente, sin poder hacer pruebas con los ficheros originales de algún programa de análisis genético de poblaciones.

Ha sido necesario cambiar algunos momentos de tecnología para llevar a cabo el proyecto pero se ha sabido solventar sin ningún problema y sin tener que cambiar el alcance del proyecto.

8.1. Lecciones aprendidas.

El desarrollo del proyecto ha sido mejor de lo que se esperaba, debido a los problemas que se tuvo con la iniciación del mismo, me ha hecho entender que con una buena planificación y un buen seguimiento y control, es posible en cierto modo ir controlando las fases del mismo.

La mayor dificultad con la que me he enfrentado es la elección de la forma de afrontar el proyecto, debido a que la primera elección no fue la correcta y tuve que cambiar la forma de afrontarlo, por ello no basta con coger lo primero que se encuentra, sino que hay que hacer un balance de todo y después con ese estudio realizado elegir la opción que más se adecue a nuestra forma de trabajar y a los objetivos del proyecto.

He aprendido a que no hay que aferrarse a lo que uno ya sabe, es bueno equivocarse y darse cuenta de lo que se utiliza no es lo correcto, y sobre todo innovar y ser creativo.

El hecho de tener que gestionar mi tiempo, yo mismo sin tener que ir a ninguna empresa a realizar el trabajo, me ha puesto a prueba, tanto en mi motivación por hacer las cosas que me propongo como mi independencia.

El proceso de lanzar la aplicación con un servidor y que sea accesible por internet ha supuesto un paso más grande en mi aprendizaje, puesto que antes no había realizado nada similar.

Al final he logrado cumplir los objetivos marcados al inicio del proyecto, y terminarlo. Estoy satisfecho con el trabajo realizado.

8.2. Propuesta de mejoras.

En cuanto al proyecto, como comentamos anteriormente dentro de los objetivos, había algunas mejoras que ofreció el cliente para llevar a cabo, si se podían realizar dentro del límite de tiempo establecido. Como no se han podido llevar a cabo se han dejado para una mejora en las próximas versiones de la aplicación.

Las mejoras son las siguientes:

- Hacer un pequeño formulario para poder obtener información adicional por parte del usuario, que en el formato de entrada elegida de no tiene, y que le sería útil tener esa información en el formato de salida.
- Implementación de las demás herramientas, debido a que solo se han implementado la transformación de 3 herramientas.

Bibliografía.

- Dudas sobre problemas producidos. https://stackoverflow.com/
- Librerías a utilizar y su funcionamiento. http://www.oracle.com
- Tutorial sobre JSP. https://www.javatpoint.com/jsp-tutorial
- Ejemplo de CSS. https://codepen.io/ehermanson/pen/KwKWEv
- Ejemplos sobre la utilización de XSLT en Servlets. http://programacion.net/
- Modificar ficheros XML. http://www.journaldev.com
- Utilización de Xjc. https://docs.oracle.com/